

# R programming

Samvel B. Gasparyan

Version Date: 2018-11-30 08:05:47, R version 3.5.0 (2018-04-23)

## Contents

<b>1</b>	<b>Ներածություն</b>	<b>3</b>
1.1	Տվյալագիտությունը որպես նոր գիտություն . . . . .	3
1.1.1	Նոր գիտության առաջացումը . . . . .	3
1.1.2	Տվյալագիտության կարևոր առանձնահատկությունները . . . . .	3
1.1.3	Եզրափակում . . . . .	5
1.2	R լեզուն . . . . .	5
1.2.1	Կիրառելիությունը . . . . .	5
1.2.2	Սրտեղծման պարամետրները և հիմնական հատկությունները . . . . .	5
1.2.3	Թողարկումները . . . . .	6
1.2.4	Ամփոփում . . . . .	6
<b>2</b>	<b>R ծրագրավորման միջավայր</b>	<b>7</b>
2.1	Լեզվի շարահյուսությունը (Syntax) . . . . .	7
2.1.1	Թվաբանական գործողությունները . . . . .	7
2.1.2	Վերագրում . . . . .	8
2.1.3	Փոփոխականներ . . . . .	9
2.1.4	Ֆունկցիաներ . . . . .	9
2.1.5	Ակտիվացնել գրադարաններ . . . . .	11
2.1.6	Ամփոփում . . . . .	11
2.2	Տվյալների տեսակները և տվյալային կառուցվածքներ . . . . .	12
2.2.1	Տվյալների տեսակները (Data Types) . . . . .	12
2.2.2	Նամասեռ վեկտորներ (Atomic Vectors) . . . . .	14
2.2.3	Գործողություններ վեկտորների հետ . . . . .	16
2.2.4	Տեսակի փոփոխություն (Type Coercion) . . . . .	17
2.2.5	Բացակայող արժեքներ (Missing Values) . . . . .	19
2.2.6	Ենթավեկտոր (Subsetting a Vector) . . . . .	21
2.2.7	Օբյեկտների հատկանիշեր (Attributes) . . . . .	26
2.2.8	Մատրիցներ (Matrices) . . . . .	27
2.2.9	Գործողություններ մատրիցների հետ . . . . .	30
2.2.10	Ենթամատրից (Subsetting Matrices) . . . . .	33
2.2.11	Ցուցակներ (Lists) . . . . .	35
2.2.12	Ֆակտորներ (Factors) . . . . .	36
2.2.13	Ուղղանկյունաձև տվյալների կառուցվածք (Data Frames) . . . . .	38
2.2.14	Ենթակառուցվածքներ (Subsetting) . . . . .	40
2.2.15	Ենթակառուցվածք ուղղանկյուն-տվյալներից (Subsetting a Data Frame) . . . . .	44
2.2.16	Կառուցվածքի փոփոխություն . . . . .	47
2.2.17	Նարգեր . . . . .	49
2.2.18	Խնդիրներ . . . . .	51
2.2.19	Ամփոփում . . . . .	54
2.3	Կարգավորման կառուցվածքներ (Control Structures) . . . . .	55
2.3.1	Տրամաբանություն (Logic) . . . . .	55
2.3.2	Գործողությունների իրականացման հերթականության կարգավորում (Control Flow) . . . . .	56
2.3.3	Շրջապտույտի կառուցվածքներ (Loop Structures) . . . . .	57
2.3.4	Repeat, Next, Break, Return . . . . .	60
2.3.5	Խնդիրներ . . . . .	61

2.3.6	Ամփոփում . . . . .	62
2.4	Ֆունկցիաներ . . . . .	62
2.4.1	Ֆունկցիայի սահմանումը . . . . .	62
2.4.2	Նրանանի կատարում ըստ անհրաժեշտության (Lazy Evaluation) . . . . .	65
2.4.3	Բազմակերպարգումները (Ellipsis or dot-dot-dot) . . . . .	66
2.4.4	Մխալների վերհանումը ֆունկցիայում (Debugging) . . . . .	68
2.4.5	Արժեքների կցում (Symbol Binding) . . . . .	70
2.4.6	Ներդրված ֆունկցիաներ . . . . .	70
2.4.7	Տեսանելիության շրջանակ (Scoping Rules) . . . . .	71
2.4.8	Վերագրման գործողություններ . . . . .	76
2.4.9	Խնդիրներ . . . . .	79
2.4.10	Ամփոփում . . . . .	80
2.5	Split - Apply - Combine . . . . .	81
2.5.1	lapply(), sapply() . . . . .	81
2.5.2	apply() . . . . .	84
2.5.3	mapply() . . . . .	84
2.5.4	vapply(), tapply() . . . . .	85
2.5.5	split() . . . . .	86
2.5.6	Խնդիրներ . . . . .	87
2.5.7	Ամփոփում . . . . .	91
2.6	Տվյալների հետ աշխատանք . . . . .	91
2.6.1	Ամսաթիվ և ժամանակային տվյալներ . . . . .	91
2.6.2	Տեքստային տվյալների հետ աշխատանք . . . . .	95
2.6.3	Տվյալների կարդացումը համակարգչից և գրումը համակարգչում . . . . .	97
2.6.4	Խնդիրներ . . . . .	99
2.6.5	Ամփոփում . . . . .	99
2.7	Կիրառական վիճակագրություն . . . . .	100
2.7.1	Սկզբնական վիճակագրական գաղափարներ . . . . .	100
2.7.2	Նիմնական գծապատկերների տեսակները . . . . .	100
2.7.3	Խնդիրներ . . . . .	100
2.7.4	Ամփոփում . . . . .	100
3	Տվյալագիտական եզրույթներ . . . . .	100

## List of Tables

## List of Figures

2.1	Ենթամադրից . . . . .	34
2.2	Շրջապտույտի ֆունկցիաներ . . . . .	81

# 1 Ներածություն

## 1.1 Տվյալագիտությունը որպես նոր գիտություն

Տիմոթի Է. Դոնոհո, “50 years of Data Science” հոդվածի վրա:

### 1.1.1 Նոր գիտության առաջացումը

Լայնորեն փարածում գրած ժամանակակից Տվյալագիտություն առարկան (Data Science) իր սկիզբն է առնում 1962թ.-ին Ջոն Թուկեյի (John Tukey) հոդվածով, որում նա կոչ էր անում ռեֆորմացիայի ենթարկել համալսարանական վիճակագրությունը և մարմնացույց էր անում նոր գիտության առաջացումը, որի ուսումնասիրության առարկան տվյալներից սովորելն է կամ տվյալների վերլուծությունը: Տվյալագիտություն եզրույթը ավելի ուշ է ներմուծվել՝ Բիլլ Բլեյվենդի կողմից (Bill Cleveland): Տվյալագիտության առաջացումը որպես գիտություն, որը զբաղվում է տվյալների հավաքագրմամբ և վերլուծմամբ, ստեղծում է բնական հարց թե ինչո՞վ է այն փարբերվում վիճակագրությունից, մանավանդ կիրառական վիճակագրությունից: Կիրառական վիճակագրությունը իր ստեղծման օրերից զբաղվել է նման հարցերով և նոր գիտության առաջացման իմաստն անորոշ է: Այս նորաստեղծ գիտությունը հաճախ ներկայացվում է որպես վիճակագրության և մեթենայական ուսուցման միավորում, որտեղ վերջինն իր հետ բերում է նաև մեծ տվյալների հետ աշխատելու գործիքակազմ: Նման սահմանումը չի արտահայտում այն մրավոր նորարարությունը, որ ընկած է այս նոր գիտության հիմքում: Քանզի գիտությունն առհասարակ շուրջով վերածվելու է տվյալների որոնք կարելի է վերլուծել, տվյալագիտության բերած նորամուծությունը ոչ միայն տվյալների մեծ ծավալի հետ է կապված, այլ՝ առհասարակ գիտության մեջ տվյալների վերլուծության գիտական հետազոտությունների (scientific studies) առաջացմամբ: Ներկայումս տվյալագիտության մեջ ցանկացած մեթոդական նորամուծություն իր ազդեցությունն է ունենալու առհասարակ գիտության զարգացման վրա: Այսպիսով՝ տվյալագիտության առաջացումը նպաստել է փաստերի վրա հիմնված (evidence based) գիտության առաջացմանը:

Ինքը Թուկեյն, իր ելակերպային հոդվածում այսպես է ընդգծում տվյալագիտության և կիրառական վիճակագրության փարբերությունը. տվյալագիտությունը նոր գիտություն է, ոչ թե մաթեմատիկայի նոր ճյուղ: Գիտություն համարվելու համար երեք բաղադրիչների առկայությունն է պարտադիր.

1. ինտելեկտուալ բովանդակությունը
2. հասկանալի կառուցվածքը
3. արդյունքների վերջնական հաստատումը փեղի է ունենում փորձի հիման վրա:

Ըստ այս սահմանման մաթեմատիկական գիտություն չէ, քանի որ նրանում վերջնական հաստատման չափանիշը փրամաբանական ամբողջականության և ապացուցելիության համաձայնեցված փեսակն է: Տվյալագիտությունը բավարարում է այս երեք պայմաններին, հետևաբար այն գիտություն է, որը բնորոշվում է ոչ թե հափուկ ուսումնասիրության առարկայով, այլ ամենուրեք առկա խնդիրներով:

Թուկեյն նաև առանձնացրել է չորս ուժեր որոնք առաջ են մղում նոր գիտությանը՝

1. վիճակագրական փետությունները,
2. արագացող զարգացումները համակարգիչներում և ցուցադրող սարքերում,
3. փարբեր ոլորտներում առկա անընդհար աճող տվյալների զանգվածը,
4. ավելի ու ավելի շար բնագավառներում քանակական արդյունքների վրա շեշտադրումը:

Կարևոր է նկատել, որ այս հարկանիշների թվարկումից ստացվում է, որ վիճակագրությունը մասամբ է ներառված տվյալագիտությունում:

### 1.1.2 Տվյալագիտության կարևոր առանձնահատկությունները

#### 1.1.2.1 Նաշվողական ծրագրային միջավայրներ՝ **Quantitative programming environments**

Վերջին 50 տարիների ընթացքում տվյալների վերլուծության համար ստեղծվել են բավականաչափ հաշվողական ծրագրային միջավայրեր: Նամալսարանական վիճակագրությունում գերիշխող դիրք ունի

Ք լեզուն: Այն սկրիպտային (script) լեզու է, որում գործողությունները կատարվում են փող առ փող՝ ճշգրտորեն համապատասխանելով հաշվողական քայլերի հերթականությանը: Ք-ի կատարած հեղափոխության կարևորագույն առանձնահատկություններից է փաթեթային ծրագրերի (code) փարածումը համացանցում և դրանց վերաօգտագործումը այլ օգտատերերի կողմից: Այս գործընթացը հեշտացվում է նաև վերը նշված առանձնահատկության պարճառով, այն է՝ քայլերի ասփիճանական կատարման, հեղափոխություն փոխելով այլ օգտատերի փաթեթային ծրագրի մեկ կամ մի քանի փողը հնարավոր է այն հարմարեցնել ձեր նպատակներին: Այսպիսով՝ հնարավոր է անընդհատ բարելավել խնդիրների ծրագրային լուծումը, սրելով վերափոխված ծրագրերում որոշակի գիտելիքներ և փոփոխությունների վերլուծության ծրագրային կիրառելիությունը դնել գիտական հիմքերի վրա:

### 1.1.2.2 Կանխատեսում և վիճակագրական եզրահանգումներ

Նամալսարանական վիճակագրությունը կարելի է բաժանել երկու մասի՝ վիճակագրական եզրահանգումներ և կանխատեսում (Statistical inference and Prediction): Վիճակագրական եզրահանգումներ կատարելիս վիճակագիրները նախ փորձում են համապատասխան մոդել ընտրել, որը լավագույնս կբնութագրի գոյություն ունեցող փոփոխությունները, ապա, ընտրված մոդելի հիման վրա կատարվում են եզրահանգումներ անհայտ մեծությունների վերաբերյալ: Սա համալսարանական վիճակագիրների գերակշռող զբաղմունքն է: Տվյալագիտության հաջորդ կարևորագույն առանձնահատկությունը համալսարանական վիճակագրության երկրորդ և ավելի քիչ օգտագործվող մասի՝ կանխատեսման վրա շեշտադրում կատարելն է: Կանխատեսման խնդիրներ լուծելիս մոդելի ընտրություն չի կատարվում, այլ փորձվում են բազմաթիվ կանխատեսման ալգորիթմներ և առաջնայնությունը տրվում է կանխատեսման լավագույն ճշգրտություն ունեցող ալգորիթմին: Կանխատեսման վրա ավելի մեծ շեշտադրումն էլ նպաստել է մեքենայական ուսուցում առարկայի առաջացմանը:

### 1.1.2.3 Մեքենայական ուսուցման հիմնական պարադիգմը՝ CTF (The Common Task Framework)

Կարդալ David Donoho, “50 years of Data Science“, Section 6 - The Predictive Culture’s Secret Sauce:

### 1.1.2.4 Տվյալագիտության ուսումնասիրության ամբողջ շրջանակը

Տվյալագիտության ուսումնասիրության ամբողջ շրջանակը կարելի է բաժանել 6 մասի՝

#### 1. Տվյալների ուսումնասիրություն և նախապատրաստում (Data Exploration and Preparation)

Առաջին քայլն է փոփոխությունների հետ աշխատելիս: Նավաբարձված փոփոխությունները նախ պետք է ուսումնասիրվեն պարզելու համար դրանց հետ կապված խնդիրները: Օրինակ, եթե հավաքագրվել են փոփոխություն մարդկանց վերաբերյալ, ապա պետք է ստուգել, որ մարդկանց սեռը ցույց տալով փոփոխությունների մեջ լինեն միայն երկու հնարավոր արժեքներ, կամ մարդկանց փաթեթը ցույց տալով փոփոխությունների մեջ բացասական արժեքներ չլինեն: Տվյալների սկզբնական ուսումնասիրությունից հետո սկսվում է փոփոխությունների նախապատրաստությունը վերլուծության համար: Նախապատրաստության ժամանակ կարող են կատարվել ինչպես փոփոխությունների մաքրման աշխատանքներ՝ հեռացնելու փոփոխություններում բացահայտված խնդիրները, այնպես և կատարվել փոփոխությունների խմբավորման կամ բաժանման գործողություններ՝ դրանք վերլուծության համար ավելի հարմար սարքելու նպատակով:

#### 2. Տվյալների ներկայացում և ձևափոխում (Data Representation and Transformation)

Այս փուլը հիմնականում ընդգրկում է փոփոխությունների փաթեթային ձևաչափերով ու զանազան կառուցվածքներով:

Գոյություն ունեն փոփոխությունների բաշխման ինչպես համընդհանուր սկզբունքներ և փոփոխություններ՝ ինչպես, օրինակ, հարաբերական փոփոխությունների բազաները (Relational databases) այնպես և փոփոխությունների սրացման ոլորտից կախված առանձնահատկություններ (փոփոխությունների սրացման եղանակները՝ պարկերներով սրացված փոփոխություն կամ փաթեթային փոփոխություններ՝ քաղված համացանցից): Այս փուլում փոփոխությունները պետք է կատարի փոփոխությունների կառուցվածքային վերափոխումներ՝ սկզբնական հավաքագրված փոփոխությունները նոր՝ ավելի բացահայտող փաթեթ:

#### 3. Նաշվարկներ փոփոխությունների հետ (Computing with Data)

Տվյալների վերլուծության ժամանակ տվյալագերն անպայմանորեն ստիպված է լինում կատարելու բարդ հաշվարկներ պարունակող հսկայածավալ պրոյեկտներ, այդ պարճառով հաշվողական արդյունավետության հետ կապված հարցերը պետք է ուշադրության կենտրոնում լինեն:

#### 4. Տվյալների մոդելավորում Data Modeling

Վիճակագրական եզրահանգումներում կատարվում է հավանականային մոդելի ընտրություն, իսկ կանխատեսման մեջ, կամ, նույնն է թե՛ մեքենայական ուսուցման մեջ, կատարվում է կանխատեսման ալգորիթմի ընտրություն:

#### 5. Տվյալների պարկերում և ներկայացում (Data Visualization and Presentation)

Պարզագույն դեպքում սա տվյալների պարկերային ներակայացումն է գոյություն ունեցող գծապարկերների քտեսակների միջոցով, օրինակ, սյունապարկերների, բայց հաճախ անհրաժեշտ է լինում գծապարկերների վրա կատարել խնդրի առանձնահատկություններից կախված ոճային փոփոխություններ, օրինակ, գունավորման ընտրություն: Բայց հաճախ անհրաժեշտություն է առաջանում ստեղծել տվյալների պարկերային ներկայացման նոր մեթոդներ, գծապարկերների նոր քտեսակներ:

#### 6. Գիտություն տվյալագիտության վերաբերյալ (Science about Data Science)

Տվյալների մշակման վերը նշված փուլերը կատարելիս և լուծումների հետ կապված առանձնահատկությունները գիտակցելիս պետք է նաև մտապահել, որ տվյալագիտությունը գիտություն է և մոտեցումները համընդհանուր են՝ անկախ տվյալների առաջացման ոլորտից: Ներկայումս վերը նշված փուլերի վերաբերյալ համընդհանուր գիտելիքները պետք է վեր հանվեն, քննարկվեն և փոխանցվեն՝ վերաօգտագործման ու հետագա զարգացման համար:

### 1.1.3 Եզրափակում

Տվյալների վերլուծության կարևորության աճին զուգահեռ ծրագրավորողների կողմից փորձ էր կատարվում այդ ճյուղն առանձնացնել վիճակագրությունից: Մեքենայական ուսուցման առաջացումն էլ երբեմն հակադրվում էր դասական, համալսարանական վիճակագրությանը:

Վերը նշվածը ցույց է տալիս ինչպես նման հակադրության անհրաժեշտության բացակայությունը, այնպես էլ տվյալագիտության լիակատար ինքնուրույնությունը՝ վիճակագրությունից և հաշվողական ծրագրավորումից:

## 1.2 R լեզուն

### 1.2.1 Կիրառելիությունը

R լեզուն հարուկ նպատակ ունեցող ծրագրավորման լեզու է. այն նախատեսված է տվյալների հետ աշխատելու համար: Այն միակ նմանատիպ լեզուն չէ, այլ օրինակներից են Python և SAS լեզուները, և տվյալների վերլուծության ոչ բոլոր խնդիրների համար է այն լավագույն ընտրությունը: Այն բավականաչափ հարմար է գծագրեր գծելիս, տվյալներ մշակելիս և վիճակագրական մոդելներ կառուցելիս, դժվարությունների առաջ կարող է կանգնել, օրինակ, համակարգի օպերատիվ հիշողությունում չտեղավորվող տվյալների հետ աշխատելիս: Այդ պարճառով հաճախ այն օգտագործվում է այլ ծրագրերի հետ համատեղ, ինչպես, օրինակ, մեծ տվյալների հետ աշխատելիս կարող են օգտագործվել տվյալների բազաների կառավարման համակարգեր, ինչպիսին են MySQL, PostgreSQL, SQLite կամ Oracle-ը:

### 1.2.2 Ստեղծման պատմությունը և հիմնական հատկությունները

R ծրագրավորման լեզուն ծագել է S ծրագրավորման լեզվից, այդ պարճառով համարվում է վերջինիս բարբառ: S ստեղծվել է 1976թ.-ին՝ Ջոն Չեմբերսի (John Chambers) կողմից և սկզբնական շրջանում հանդիսանում էր Fortran լեզվի գրադարանների հավաքածու: 1988թ.-ին այն ամբողջովին վերափոխվում է և գրվում C լեզվով ու ստանում ժամանակակից տեսք: Սա լեզվի երրորդ թողարկումն էր (S3): 1998-ին թողարկվում է լեզվի չորրորդ տարբերակը:

R-ը ստեղծվել է Նոր Զելանդիայում 1991թ.-ին՝ Ռոսս Իհակայի և Ռոբերտ Զենթլմենի կողմից (Ross Ihaka and Robert Gentleman): 2000-ին լույս է տեսնում լեզվի 1.0.0 թողարկումը, իսկ 2018-ին՝ 3.5.0 թողարկումը:

R լեզուն ազատ օգտագործման իրավունքով է փարածվում (open source software): Սա նշանակում է, որ այն անվճար է փարածվում նաև փրեդի ունեն հեղինակ պայմանները՝

- ծրագիրն օգտագործելու ազատություն՝ անկախ նպատակից,
- ազատություն ուսումնասիրելու, թե ինչպես է ծրագիրն աշխատում: Մրա նախապայման է հանդիսանում ծրագրերի ներքին կառուցվածքին հասանելիությունը,
- յուրաքանչյուր ոք իրավունք ունի այն փարածելու իր հայեցողությամբ,
- ազատություն բարելավելու ծրագիրը և ազատ փարածելու բարելավումները:

Սա նաև նշանակում է, որ լեզուն ունի բազմաթիվ գրադարաններ (packages) գրված աշխարհի փարբեր անկյուններում ապրող ծրագրավորողների կողմից: Ցանկացած ոք իրավունք ունի ստեղծելու նոր գրադարան և այն փրեդադրելու կենտրոնական գրադարանում, որը կոչվում է՝ CRAN (The Comprehensive R Archive Network), այդպիսով այն հասանելի դարձնելով լեզվի բոլոր օգտագործողներին: Որպեսզի օգտագործողի ստեղծած գրադարանը հնարավոր լինի փրեդադրել կենտրոնական գրադարանում, այն պետք է որոշակի ստանդարտների բավարարի, օրինակ ունենա համապատասխան օգտագործման ուղեցույց (package documentation):

R լեզուն «երկխոսող» (interactive) է՝ այն կարող է ստանալ հարցն ու միանգամից դրան պատասխանել, այնուհետև լսել նոր հարց: Անգամ ծրագիրը, որը բաղկացած է մի քանի փրոդերից՝ կարդացվում է փող առ փող: Այս պարճառով R-ն ունի աշխատելու երկու վիճակ՝ երկխոսության վիճակ, երբ հրամանները կատարվում են փող առ փող ներմուծելիս՝ փրոդային հրամանների համար նախատեսված պատուհանում, (console mode) կամ փրեքսպային պատուհանում գրելով մի քանի փրոդից բաղկացած ծրագիր և այն ամբողջությամբ ներմուծելիս (հիշենք, որ նույնիսկ այս դեպքում ծրագիրը կարդացվում է փող առ փող:) Այս վերջինը կոչվում է՝ script mode:

### 1.2.3 Թողարկումները


R լեզուն ունենում է նոր թողարկումներ փարեկան մոփավորապես երկու անգամ: Նոր թողարկումների ժամանակ ավելացվում են լեզվի նոր գործառնություններ, շրվվում առկա թերությունները, երբեմն նաև կատարվում են հին գործառնությունների փոփոխություններ: Ներառաբար կարևոր է ծրագիր գրելիս նշել այն թողարկման համարը, որով այն գրված է: Երբեմն նոր թողարկումներով հին ծրագիրն աշխատեցնելիս կարող են ի հայտ գալ անսպասելի արդյունքներ, կամ պարզապես սխալ փրեդի ունենա: Նշվածը վերաբերվում է նաև ծրագրում օգտագործված բոլոր գրադարաններին: Այդ պարճառով այս դասախոսությունում մենք միշտ կնշենք լեզվի և գրադարանների թողարկման համարները, որոնք օգտագործվել են դասախոսությունում առկա ծրագրերը գրելիս:

```
## [1] "R version 3.5.0 (2018-04-23)"
```

```
## [1] "Joy in Playing"
```


### 1.2.4 Ամփոփում

- CRAN
- Evidence based science
- Scientific studies
- Quantitative programming environment
- Code (փրեքսպային ծրագիր)
- Open source software
- Packages
- Interactive programming

- Console mode
- Script mode
-  version
- Package version
- Package documentation
- Statistical inference (Վիճակագրական եզրահանգումներ)
- Prediction (Կանխատեսում)
- CTF - The common task framework
- Data science, data scientist (Տվյալագիտություն, ավյալագետ)
- Graph (գծապատկեր)
- Visualization (Պատկերավորում, պատկերում, պատկերագրում)

## 2 R ծրագրավորման միջավայր

### 2.1 Լեզվի շարահյուսությունը (Syntax)

 Ծրագիրը բաղկացած է հրամաններից, որոնք, ինչպես նշեցինք վերևում, կարդացվում են փող առ փող, իսկ մեկ փողի վրա հնարավոր է մի քանի հրամաններ գրել՝ բաժանելով դրանք “;” նշանով: Ներկայումս նշանը հանդիսանում է մեկնաբանությունների նշան և դրանից ձախ գրված ամեն ինչ, այդ թվում նշանն ինքը, հանդիսանում է մեկնաբանություն և անտեսվում է ծրագրի կողմից՝

```
# Comments that are ignored
1+6
```

```
## [1] 7
```

Նրամանները պետք է շարահյուսորեն ճիշտ լինեն կազմված որպեսզի կարողանան կախարվել ծրագրի կողմից: Օրինակ հերկյալ հրամանը շարահյուսորեն թերի է և չի կախարվի ծրագրի կողմից՝


```
6+
```

Երբ ներմուծվում է թերի հրաման, ապա փողային հրամանների համար նախատեսված պատուհանում (console mode) ծրագիրը պատրաստ է հրամանների կախարման նշանը “>” փոխարինվում է սպասողական նշանով “+”, որի դեպքում կան պետք է լրացնել հրամանը, որ այն թերի չլինի, կան էլ սեղմել ելքի սրբղնը՝ Esc:


Վերևում նկարագրված թվաբանական գործողությունը ներմուծելիս մենք իրականում ծրագրին փախի ենք երկու գործողություններ՝ կախարման համար, նախ կախարել թվաբանական գործողությունը, ապա այն փախել: Սա կոչվում է ինքնաբերաբար փախում (auto-printing), երբ մենք բացահայտ չենք կանչում `print()` ֆունկցիան՝

```
print(6+1)
```

```
## [1] 7
```

 - ում մեծապատերը և փոքրապատերը փառքերակվում են (case sensitive), այդ պատճառով անուններում գոնե մեկ փառք մեծապատով գրելու դեպքում այն դառնում է լրիվ այլ անուն:

#### 2.1.1 Թվաբանական գործողությունները

-ում առկա են հերկյալ թվաբանական գործողությունները՝

```
4-6
```

```
## [1] -2
```

```
4+6
```

```
## [1] 10
```

```
2*3
```

```
## [1] 6
```

```
2/3
```

```
## [1] 0.6666667
```

```
2^3
```

```
## [1] 8
```

```
3%%2
```

```
## [1] 1
```

```
3%/%2
```

```
## [1] 1
```

Գործողությունների առաջնահերթությունը (operator precedence, operator priority) հետևյալն է՝ նախ կատարվում է ասպիճանի բարձրագույն գործողությունը, ապա բազմապատկում, բաժանում, մնացորդի գրնում, կամ առանց մնացորդի բաժանումը գործողությունը, իսկ վերջում՝ գումարում կամ հանումը:

```
3+10*2
```

```
## [1] 23
```

```
3*2^2
```

```
## [1] 12
```

```
5%%2*3
```

```
## [1] 3
```

Նավասարագոր գործողությունների կատարման հերթականությունը որոշվում է ձախից աջ: Օրինակ,

```
10/5/2
```


```
## [1] 1
```

Իհարկե հնարավոր է փոխել գործողությունների կատարման առաջնահերթությունը՝ օգտագործելով փակագծեր:

```
(2+3)^2
```

```
## [1] 25
```

### 2.1.2 Վերագրում

Վերագրման գործողությունը կատարվում է <- նշանի միջոցով, որը սլաք է և ցույց է տալիս վերագրման ուղղությունը՝ աջից ձախ: Այլ ծրագրավորման լեզուներում վերագրում կատարվում է հավասարության նշանի (=) միջոցով: Նավասարության նշանը  լեզվում նույնպես առկա է, բայց մի քիչ այլ գործառնությ ունի: Նաճախ գրքերում կարելի է տեսնել, որ վերագրում կատարված է հավասարության նշանի միջոցով և իրականում եթե ձեր ծրագրում օգտագործեք հավասարության նշանը, ապա դեպքերի գերակշիռ մեծամասնությունում որևէ խնդրի չեք հանդիպի: Բայց սլաքի և հավասարության նշանի միջև կա տարբերություն և մենք հետագայում օրինակների



վրա կհանդգնենք: Առայժմ միայն նկատենք, որ ի փարբերություն հավասարություն նշանի՝ սլաքը թույլ է փալիս նաև վերագրում իրականացնել նաև ձախից աջ՝

```
x<-5
x
```

```
## [1] 5
```

```
6->x
x #print(x)
```

```
## [1] 6
```

Այսպեղ նույնպես փեղի ունի ինքնաբերաբար փալումը, երբ մենք ներմուծում ենք միայն փոփոխականի անունը:

### 2.1.3 Փոփոխականներ

Փոփոխական սրեղծելիս նրա փեսակը նախապես չի հայտարարվում: Վերագրման գործողությունը կափարելիս ծրագիրն ինքը կարող է կռահել փոփոխականի փեսակը: Այս պափճառով հնարավոր է անգամ գոյություն ունեցող փոփոխականին վերագրել այլ փեսակի արժեք, այդ դեպքում հին արժեքն իր փեսակով կջնջվի և փոփոխականում կպահվի նոր արժեքն իր փեսակով: Փոփոխականին հնարավոր է վերագրել ցանկացած օբյեկտ, օրինակ, ֆունկցիա (փեն հաջորդ ենթաբաժնում):

**R**-ում կան սահմանափակումներ փոփոխականի անուններ ընփրելիս: Փոփոխականի անունը կարող է պարունակել թվեր, փառեր, կեփ կամ փողի ներքևում գրվող գծիկ՝ —: Անունը կարող է սկսվել միայն փառով կամ կեփով (այս դեպքում հաջորդը չի կարող թիվ լինել), բացի այդ, ինչպես նշվեց, **R**-ը փարբերակում է մեծափառերը և փոքրափառերը, հեփնաբար պեփք է ուշադիր լինել անուն ընփրելիս:

### 2.1.4 Ֆունկցիաներ

**R**-ում ֆունկցիաները կափարում են նույն գործառույթը ինչ մաթեմափիկայում՝ նրանք սրեղծում են համապափախանույթուն օբյեկտների միջև:

Վերցնելով որոշ օբյեկտ, ֆունկցիան կափարում է գործողություններ դրա հեփ ու վերադարձնում այլ օբյեկտ: Սկզբնական օբյեկտը կոչվում է ֆունկցիայի արգումենտ: **R**-ը պարունակում է բազմաթիվ նախասահմանված ֆունկցիաներ (built-in functions), որոնք նախափեսված են փարբեր փեսակ խնդիրներ լուծելու համար: Նախասահմանված ֆունկցիաների ամենապարզ օրինակ են հանդիսանում թվաբանական գործողությունները և վերագրումը՝

```
`<-` (u, 2018)
u
```

```
## [1] 2018
```

```
5+19
```

```
## [1] 24
```

```
`+` (5, 19)
```

```
## [1] 24
```

Մաթեմափիկական գործողությունների համար ևս կան նախասահմանված ֆունկցիաներ՝

```
sqrt(16)
```

```
## [1] 4
```

```
abs(-5)
```

```
## [1] 5
```

Քանի որ ամեն ինչ օբյեկտ է, ապա հնարավոր է փոփոխականին վերագրել անգամ ֆունկցիաներ՝

```
x<-`+`  
x(12,34)
```

```
## [1] 46
```

**R**-ում նախասահմանված ֆունկցիաներ կան, որոնք կարող են փոփոխականներին փոխարինել օգտագործողի կողմից սահմանված փոփոխականների վերաբերյալ՝

```
ls()
```

```
## [1] "u" "x"
```

Ինչպես նաև, հնարավորություն ընձեռել ջնջելու սահմանված փոփոխականներից մեկը կամ մի քանիսը, ինչպես նաև բոլորը՝

```
ls()
```

```
## [1] "u" "x"
```

```
rm(x)  
ls()
```

```
## [1] "u"
```

```
x<-10;y<-12;z<-13;u<-54  
ls()
```

```
## [1] "u" "x" "y" "z"
```

```
rm(x,y)  
ls()
```

```
## [1] "u" "z"
```

Գոյություն ունեն ծրագրավորման որոշակի կանոններ, որոնց օգտագործումը թույլ կտա խուսափել բազմաթիվ հնարավոր սխալներից (Good Programming Practice): Այդ սկզբունքներից է՝ ամեն նոր ծրագրավորման խնդիր լուծելիս ջնջել բոլոր սրբեղծված օբյեկտները, որոնք մնացել էին նախորդ ծրագիրն իրականացնելիս: Դա կարող է կատարվել հետևյալ ձևով՝

```
rm(list=ls())  
ls()
```

```
## character(0)
```

Կան նաև բազմաթիվ նախասահմանված ֆունկցիաներ որոնք փոփոխականներ են փոխարինում **R** թողարկման մասին, ինչպես օրինակ

```
sessionInfo()
```

```
## R version 3.5.0 (2018-04-23)  
## Platform: x86_64-w64-mingw32/x64 (64-bit)  
## Running under: Windows 10 x64 (build 15063)  
##  
## Matrix products: default  
##  
## locale:  
## [1] LC_COLLATE=English_United States.1252  
## [2] LC_CTYPE=English_United States.1252  
## [3] LC_MONETARY=English_United States.1252  
## [4] LC_NUMERIC=C  
## [5] LC_TIME=English_United States.1252
```

```
##
## attached base packages:
## [1] stats      graphics  grDevices  utils      datasets  methods   base
##
## loaded via a namespace (and not attached):
## [1] compiler_3.5.0  backports_1.1.2 magrittr_1.5    rprojroot_1.3-2
## [5] tools_3.5.0     htmltools_0.3.6 yaml_2.1.19     Rcpp_0.12.17
## [9] stringi_1.1.7   rmarkdown_1.9  knitr_1.20      stringr_1.3.1
## [13] digest_0.6.15  evaluate_0.10.1
```

**R**-ում նախասահմանված բոլոր ֆունկցիաները խմբավորված են նախասահմանված հիմնական գրադարաններից որևէ մեկում (base packages), որոնք յոթն են և միշտ ակտիվացված են (attached packages):

### 2.1.5 Ակտիվացնել գրադարաններ

```
#install.packages("HistData")
library(HistData)
```

```
## Warning: package 'HistData' was built under R version 3.5.1
```

### 2.1.6 Ամփոփում


- Assignment
- Syntax
- Built-in functions
- Auto-printing
- Case sensitive
- Attached packages
- Base packages
- Good Programming Practice

#### Functions


- *print()*
- *sqrt()*
- *abs()*
- *ls()*
- *rm()*
- *sessionInfo()*
- *library()*
- *install.packages()*
- *library()*

## 2.2 Տվյալների տեսակները և փվյալային կառուցվածքներ

### 2.2.1 Տվյալների տեսակները (Data Types)

-ում կան փվյալների պահպանման 5 տեսակներ՝

- character - փեքսպային
- numeric (real numbers) - թվային (իրական թվեր)
- integer - ամբողջ թվեր
- complex - կոմպլեքս թվեր
- logical (True/False) - փրամաբանական՝ այո, ոչ

 - ում, ի փարբերություն շար ծրագրավորման լեզուների, փոփոխականը սպեղծելիս նրա տեսակը չի հայտարարվում: Փոփոխական հնարավոր է սպեղծել նրան արժեք վերագրելով և ծրագիրն ինքը, հիմք ընդունելով արժեքը, կսպեղծի փոփոխականի տեսակը: Նախասահմանված `class()` ֆունկցիան վերադարձնում է փոփոխականի տեսակը:

```
x<-"Name"
class(x)
```

```
## [1] "character"
```

```
x<-1
class(x)
```

```
## [1] "numeric"
```

```
x<-1L
class(x)
```

```
## [1] "integer"
```


```
x<-TRUE
class(x)
```

```
## [1] "logical"
```

```
x<-3+1i
class(x)
```

```
## [1] "complex"
```

Նկատենք, որ բոլոր իրական թվերը, այդ թվում ամբողջ թվերը ծրագրի կողմից հասկացվում են որպես numeric տեսակի, այսինքն՝ փասնորդական ճշտության իրական թվեր: Որպեսզի ծրագիր ներմուծված թիվը հասկանա որպես ամբողջ թիվ՝ վերջում պետք է ավելացնել `L` փառը:

 - ում կա նաև հատուկ թիվ, որով նշանակվում են շար մեծ թվերը և որը կապարում է մաթեմատիկայի անվերջության դերը: Այն նշանակվում է `Inf`:

```
2^1024
```

```
## [1] Inf
```

```
1/0
```

```
## [1] Inf
```

```
Inf+5
```

```
## [1] Inf
```

```
Inf*(-3)
```

```
## [1] -Inf
```

Նիշենք, որ **R**-ը տարբերակում է մեծապատերը և փոքրապատերը, հետևաբար ուշադրություն դարձնենք, որ *Inf* հատուկ անունը սկսվում է մեծապատով:

*R*-ում կա նաև *NaN* հատուկ արժեքը, որը ցույց է տալիս չսահմանված գործողությունների արդյունքում ստացված արժեքները NaN:

```
0/0
```

```
## [1] NaN
```

Տրամաբանական արժեքները մեծամասամբ ստացվում են տրամաբանական գործողությունների արդյունքում՝ համեմատում, ժխտում, և, կամ:

```
x<-3<4
x
```

```
## [1] TRUE
```

```
x<-4<=4
!x
```

```
## [1] FALSE
```

```
x==TRUE
```

```
## [1] TRUE
```

```
x&3<2
```

```
## [1] FALSE
```

```
x|3<2
```

```
## [1] TRUE
```

```
6!=7
```

```
## [1] TRUE
```

Նամենաբանական գործողությունները հնարավոր է օգտագործել նաև տեքստային տվյալների հետ աշխատելիս: Երկու բառերը համեմատելիս մենք հարցնում ենք թե առաջին բառի առաջին տառը այբբենաբանում երկրորդ բառի առաջին տառից ավելի շուրտ է հանդիպում: Եթե այո, ապա առաջին բառն ավելի փոքր է <, քան երկրորդ բառը: Եթե երկու բառն էլ սկսվում են նույն տառով, ապա համեմատվում են երկրորդ տառերը և այսպես շարունակ՝

```
"Ararat">"Armenia"
```

```
## [1] FALSE
```

Պարզագույն տեսակ պարունակող բոլոր փոփոխականների երկարությունը 1 է, անգամ եթե փոփոխականի տեսակը տեքստային է և այն կազմված է բազմաթիվ տառերից: Երկարությունը ստացվում է *length()* ֆունկցիայի միջոցով, իսկ տեքստում առկա նշանների քանակը՝ *nchar()* ֆունկցիայի միջոցով՝

```
x<-12
length(x)
```

```
## [1] 1
```

```
x<-"This is a very long sentence"
length(x)
```

```
## [1] 1
```

```
nchar(x)
```

```
## [1] 28
```

Մաթեմատիկայից մեզ հայտնի է, որ  $-1$  թվի արմատը կոմպլեքս թիվ է, սակայն պետք է  $\mathbb{R}$ -ում ուշադիր լինել քանի որ  $\text{sqrt}(-1)$  հրամանը չի աշխատում: Ըստ  $\text{sqrt}()$  ֆունկցիայի սահմանման՝ դրա վերադարձվող արժեքը ներվածված արժեքի տեսակի է, այսինքն՝  $-1$  ներմուծված թիվն իրական թիվ է և վերադարձվող թիվը նույնպես պետք է իրական լինի: Փոխարենը մենք կարող ենք կատարել՝

```
sqrt(-1+0i)
```

```
## [1] 0+1i
```

Պարզագույն տեսակների հիման վրա կառուցվում են տվյալների պահպանման ավելի բարդ օբյեկտներ, որոնք մենք կանվանենք տվյալների պահպանման կառուցվածքներ (Data structures): Մրանցից առաջինը կանվանենք համասեռ վեկտոր:

### 2.2.2 Նամասեռ վեկտորներ (Atomic Vectors)

Մաթեմատիկայում թվերի վերջավոր հաջորդականությունները կոչվում են վեկտորներ:  $\mathbb{R}$ -ում նույնպես կան վեկտորներ, որոնք ունեն նույն կառուցվածքը ինչ մաթեմատիկայում՝ փարբերությամբ, որ այսպեղ վեկտորները կարող են պահել ոչ միայն թվային տվյալներ:  $\mathbb{R}$ -ում վեկտորները կարող են ունենալ նաև բարդ կառուցվածք (օրինակ  $\mathbb{R}$ -ում մատրիցը ևս համարվում է վեկտոր), այդ պատճառով մենք ներմուծում ենք համասեռ վեկտորի գաղափարը, որը վեկտորի պարզագույն տեսակն է և համապատասխանում է մաթեմատիկայի վեկտորի գաղափարին: Նամասեռ վեկտոր ստեղծելու ամենահեշտ ձևը  $c()$  (անգլերեն combine՝ համախմբել բառի առաջին փառով նշանակվող) ֆունկցիան օգտագործելն է:

```
x<-c(1,14,5)
```

```
x
```

```
## [1] 1 14 5
```

```
class(x)
```

```
## [1] "numeric"
```

Իրականում անգամ մեկ թիվը իրենից ներկայացնում է վեկտոր՝ մեկ երկարությամբ:

Ինչպես տեսնում ենք վերևում՝ փոփոխականի տեսակը նորից թվային է վերադարձվում և միակ փարբերությունը մեկ թիվ պարունակող փոփոխականից երկարությունն է, որը կարող է ստացվել  $\text{length}()$  ֆունկցիայի միջոցով:

```
x<-c(1,14,5)
```

```
length(x)
```

```
## [1] 3
```

Նամասեռ վեկտորները կարող են պահել նաև ոչ թվային արժեքներ՝

```
x<-c(TRUE,F)
```

```
x
```

```
## [1] TRUE FALSE
```

```
class(x)
```

```
## [1] "logical"
```

```
x<-c("Name", "Surname")
```

```
x
```

```
## [1] "Name" "Surname"
```

```
class(x)
```

```
## [1] "character"
```

**R**-ում հաջորդական թվերից կազմված համասեռ վեկտոր ստեղծելու համար կարող է օգտագործվել նաև հետևյալ ֆունկցիան

```
x<-2:6 #the same as x<-c(2,3,4,5,6)
x
```

```
## [1] 2 3 4 5 6
```

Նամասեռ վեկտոր կարող է ստեղծվել նաև `vector()` ֆունկցիայի միջոցով՝

```
x<-vector(mode = "integer",length = 10)
x
```

```
## [1] 0 0 0 0 0 0 0 0 0 0
```

Ստեղծելիս ծրագիրը սկզբնական արժեքներ է փախի վեկտորին, թվային տեսակի վեկտորների դեպքում սկզբնական արժեքը 0-ն է:

Նամասեռ վեկտոր ստեղծելու մեկ այլ հնարավորություն է կրկնել միևնույն արժեքը որոշակի թվով անգամ:

```
x<-rep(3,times=4)
x
```

```
## [1] 3 3 3 3
```

```
x<-rep(c(0,1,2),times=5)
x
```

```
## [1] 0 1 2 0 1 2 0 1 2 0 1 2 0 1 2
```

```
x<-rep(c(1,3),each=3)
x
```

```
## [1] 1 1 1 3 3 3
```

Նամասեռ վեկտոր ստեղծելու հանարավորություն է նաև ``` գործողությունը:

```
1:6
```

```
## [1] 1 2 3 4 5 6
```

```
`:(1,6)
```

```
## [1] 1 2 3 4 5 6
```

```
pi:7
```

```
## [1] 3.141593 4.141593 5.141593 6.141593
```

```
7:(pi-1)
```

```
## [1] 7 6 5 4 3
```

Կամ

```
x<-seq(0,1,by=0.1)
x
```

```
## [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
x<-seq(0,1,length=5)
x
```

```
## [1] 0.00 0.25 0.50 0.75 1.00
```

Նաճախ անհրաժեշտություն է առաջանում փրված վեկտորի երկարությամբ վեկտոր ստեղծել, որը կպարունակի սկզբնական վեկտորի կոորդինատների համարակալումը: Սրա համար օգտագործվում է `seq_along()` ֆունկցիան՝

```
x<-c("First Name", "Last Name", "Position", "Country")
seq_along(x)
```

```
## [1] 1 2 3 4
```

```
#same as
1:length(x)
```

```
## [1] 1 2 3 4
```

Բացի այդ՝ երբեմն անհրաժեշտ է լինում փրված երկարությամբ վեկտոր ստեղծել, որը բաղկացած կլինի հաջորդական ամբողջ թվերից: Օգտագործվում է `seq_len()` ֆունկցիան՝

```
n<-6
seq_len(n)
```

```
## [1] 1 2 3 4 5 6
```

```
#same as
1:n
```

```
## [1] 1 2 3 4 5 6
```

Նամասեռ վեկտորների կարևորագույն հատկությունը միայն միևնույն տեսակի արժեքներ պահելու հնարավորությունն է:

### 2.2.3 Գործողություններ վեկտորների հետ

Նամասեռ թվային վեկտորների հետ հարմար է գործողությունները կատարելը: Բոլոր գործողությունները կատարվում են կոորդինատ առ կոորդինատ: Իսկ եթե վեկտորի և փարրական օբյեկտի (1 երկարությամբ վեկտորի) միջև է գործողություն կատարվում, ապա նույնպես փարրական օբյեկտի և առանձին առանձին վեկտորի կոորդինատների հետ կատարվում է այդ գործողությունը և վերադարձվում է վեկտորի երկարությամբ մեկ այլ վեկտոր՝

```
c(-1,pi,4)+1:3
```

```
## [1] 0.000000 5.141593 7.000000
```

```
c(-1,pi,4)*1:3
```

```
## [1] -1.000000 6.283185 12.000000
```

```
c(-1,pi,4)/1:3
```

```
## [1] -1.000000 1.570796 1.333333
```

```
c(-1,pi,4)^(1:3)
```

```
## [1] -1.000000 9.869604 64.000000
```

```
c(-1,pi,4)+2
```

```
## [1] 1.000000 5.141593 6.000000
```



Եթե երկու վեկտորներ ունեն փարբեր երկարություններ, ապա մինչ գործողությունը կատարելը ծրագիրը կրկնում է կարճ վեկտորն այնքան անգամ մինչ դրա երկարությունը կհավասարվի երկարին, ապա կատարում գործողությունը:

```
c(1,3,5,8)/c(2,4)
```

```
## [1] 0.50 0.75 2.50 2.00
```

Իրականում վեկտորի և փարբական փոփոխականի միջև վերը նշված գործողությունը ևս նույն սկզբունքով է կատարվում, օրինակ, երբ վեկտորը բազմապատկվում է թվով, այդ թիվը կրկնվում է վեկտորի երկարության քանակով, ապա կատարվում է գործողությունը՝

```
1:4*2
```

```
## [1] 2 4 6 8
```

```
1:4*c(2,2,2,2)
```

```
## [1] 2 4 6 8
```

Նույն սկզբունքը գործում է նաև փրամաբանական գործողություններ կատարելիս՝

```
1:10<5
```

```
## [1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE
```

Եթե երկար վեկտորի երկարությունն առանց մնացորդի չի բաժանվում կարճ վեկտորի երկարությանը, այն է՝ կարճը կրկնելով հնարավոր չէ հավասարեցնել երկարի երկարությանը, ապա գործողությունն այնուամենայնիվ կկատարվի բայց ծրագիրը կփրա նաև զգուշացում՝

```
1:5/c(1,2)
```

```
## Warning in 1:5/c(1, 2): longer object length is not a multiple of shorter
## object length
```

```
## [1] 1 1 3 2 5
```

Նամասեռ վեկտորների հետ աշխատելիս կարևոր է ստուգելու գործողությունը, թե արդյոք որևէ արժեք կամ արժեքներ այդ վեկտորի մեջ են, թե՛ ոչ:

```
"Writing"%in%c("Speaking", "Writing", "Listening", "Reading")
```

```
## [1] TRUE
```

```
c(3,5)%in%1:4
```

```
## [1] TRUE FALSE
```

```
1:4%in%c(4,3)
```

```
## [1] FALSE FALSE TRUE TRUE
```

Վերադարձվում է առաջին վեկտորի երկարության վեկտոր, որը պարունակում է փրամաբանական արժեքներ, կախված նրանից, թե արդյոք առաջին վեկտորի համապատասխան կոորդինատը պարունակվում է երկրորդ վեկտորում:

## 2.2.4 Տեսակի փոփոխություն (Type Coercion)

Ինչպես նշվեց՝ համասեռ վեկտորներում կարող են պահվել միայն միևնույն տեսակի արժեքներ: Երբ փորձենք ստեղծել համասեռ վեկտոր և դրանում պահել փարբեր տեսակի արժեքներ, ապա ոչ թե սխալ տեղի կունենա այլ՝ տեսակի հարկադիր փոփոխություն՝

```
x<-c(F,12)
x
```

```
## [1] 0 12
```

```
class(x)
```

```
## [1] "numeric"
```

```
x<-c(TRUE,1L)
x
```

```
## [1] 1 1
```

```
class(x)
```

```
## [1] "integer"
```

```
x<-c(21,"12")
x
```

```
## [1] "21" "12"
```

```
class(x)
```

```
## [1] "character"
```

**R**-ն այս դեպքերում կադարում է տեսակի ոչ բացահայտ փոփոխություն: Նմարավոր է նաև կադարել տեսակների բացահայտ փոփոխություն՝ օգտագործողի ցանկությամբ:

```
x<-0:10
class(x)
```

```
## [1] "integer"
```

```
y<-as.character(x)
y
```

```
## [1] "0" "1" "2" "3" "4" "5" "6" "7" "8" "9" "10"
```

```
y<-as.logical(x)
y
```

```
## [1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
y<-as.complex(x)
y
```

```
## [1] 0+0i 1+0i 2+0i 3+0i 4+0i 5+0i 6+0i 7+0i 8+0i 9+0i 10+0i
```

```
x<-c("1", "2", "3")
y<-as.numeric(x)
y
```

```
## [1] 1 2 3
```

Բնականաբար տեսակի փոփոխությունը ոչ միշտ է հնարավոր՝

```
x<-c("a", "b", "c")
as.logical(x)
```

```
## [1] NA NA NA
```

*NA* հերթական հատուկ արժեքն է, որը ցույց է տալիս որ տվյալ արժեքը գոյություն չունի՝ Not assigned: Տեսակի փոփոխման հերթականությունը հետևյալն է՝

Logical -> integer -> numeric -> complex -> character

### 2.2.5 Բացակայող արժեքներ (Missing Values)

Ինչպես նշվեց վերևում **R**-ում արժեքի բացակայությունը ցույց տվող արժեքներն են *NA* և *NaN* արժեքները, որոնցից վերջինը նախատեսված է չսահմանված մաթեմատիկական գործողությունները ցույց տալու համար: Որպեսզի պարզենք թե փոփոխականի մեջ պահվող արժեքը բացակայող արժեքի ինչ տիպ ունի, օգտագործվում է *is.na()* ֆունկցիան, որը վերադարձնում է փրամաբանական արժեքներ՝ այո կամ ոչ: Նմանապես գործում է նաև *is.nan()* ֆունկցիան: *NaN* տեսակի արժեք ունեցող փոփոխականը նաև *NA* տեսակի է, բայց հակառակը ճիշտ չէ:

```
x<-NA
is.na(x)
```

```
## [1] TRUE
```

```
is.nan(x)
```

```
## [1] FALSE
```

```
x<-NaN
```

```
is.na(x)
```

```
## [1] TRUE
```

```
is.nan(x)
```

```
## [1] TRUE
```

Նստուկ արժեքների հետ աշխատելիս սովորական == գործողությամբ հնարավոր չէ հավասարություն ստուգել, այդ պատճառով էլ անհրաժեշտ է օգտագործել վերը նշված ֆունկցիաները՝

```
x<-NA
x
```

```
## [1] NA
```

```
x==NA
```

```
## [1] NA
```

Արժեքի բացակայությունը ցույց տվող արժեքի՝ *NA* կարևոր առանձնահատկություններից է, որ այն հայտնվելով համասեռ վեկտորում՝ չի փոխում վեկտորի տեսակը՝ (*NaN* տեսակը թվային է, հետևաբար այն ընդգրկելով այլ տեսակի համասեռ վեկտորներում դրա տեսակը կփոխվի)

```
x<-c("1",NA,"2",NA)
class(x)
```

```
## [1] "character"
```

```
class(NaN)
```

```
## [1] "numeric"
```

```
c("1",NaN)
```

```
## [1] "1" "NaN"
```

Մրա պատճառը *NA* արժեքի որպես փրամաբանական արժեք սահմանված լինելն է, այդ պատճառով հայտնվելով այլ տեսակների համասեռ վեկտորներում այն կարողանում է փոխել իր տեսակը և դառնալ իրեն

ընդգրկող վեկտորի տեսակի (փրամաբանական տեսակը ամենավոքոր տեսակն է և այն հնարավոր է ցանկացած այլ տեսակի վերածել):

```
class(NA)
```

```
## [1] "logical"
```

Կիրառելով այս վեկտորների վրա բացակայող արժեքները հայտնաբերող ֆունկցիաները, կստանանք այն կոորդինատները, որտեղ գրված են բացակայող արժեքները՝

```
x<-c(1,NaN,2,NA)
is.na(x)
```

```
## [1] FALSE TRUE FALSE TRUE
```

```
is.nan(x)
```


```
## [1] FALSE TRUE FALSE FALSE
```

```
y<-c("1",NA,"2",NA)
class(y)
```

```
## [1] "character"
```

```
is.na(x)
```

```
## [1] FALSE TRUE FALSE TRUE
```

Այս ֆունկցիաները կիրառելիս նկատում ենք -ի կարևորագույն առանձնահատկություններից մեկը, որը հեղափոխում ավելի մանրամասն կրթնարկվի: Նշված ֆունկցիաներ որպես արգումենտ ընդունում են համասեռ վեկտոր, գործողությունը կատարում են յուրաքանչյուր կոորդինատի համար՝ առանձին առանձին, ապա վերադարձնում արժեքներից կազմված վեկտոր, որը ունի սկզբնական վեկտորին հավասար երկարություն (Vectorized function): Այս առանձնահատկության շնորհիվ շրջապատյալի ֆունկցիաների օգտագործումից (Loop) հաճախ հնարավոր է լինում խուսափել, այդպիսով՝ ավելի արագացնելով և արդյունավետ դարձնելով գործողությունների կատարումը:

Նախասահմանված `sum()` ֆունկցիան որպես արգումենտ ստանալով համասեռ թվային վեկտոր վերադարձնում է այդ վեկտորի կոորդինատների գումարը՝

```
sum(1:10)
```

```
## [1] 55
```

Այս ֆունկցիայի միջոցով հնարավոր է հաշվել համասեռ վեկտորում առկա բացակայող արժեքների՝ `NA`-ների քանակը՝

```
x<-c("A", NA, NA, NA, "B", NA, "1", NA, NA)
is.na(x)
```

```
## [1] FALSE TRUE TRUE TRUE FALSE TRUE FALSE TRUE TRUE
```

```
sum(is.na(x))
```

```
## [1] 6
```

Ինչպես նշվեց՝ `sum()` ֆունկցիան որպես արգումենտ ընդունում է համասեռ թվային վեկտորներ, հեղափոխելով դա կիրառելով համասեռ, փրամաբանական արժեքներ պարունակող վեկտորների վրա (որը ստացվում է `is.na()` ֆունկցիան կանչելու արդյունքում), տեղի է ունենում տեսակի քոդարկված փոփոխություն, ինչպես հեղափոխված ֆունկցիան կանչելիս՝

```
as.numeric(is.na(x))
```

```
## [1] 0 1 1 1 0 1 0 1 1
```

Եվ վերցնելով այս վեկտորի կոորդինատների գումարը՝ կստանանք 1-երի քանակը, իսկ վերջիններս համապարասխանում են բացակայող արժեքները ցույց փոխող *TRUE* արժեքներին:

### 2.2.6 Ենթավեկտոր (Subsetting a Vector)

**R**-ում գոյություն ունի համասեռ վեկտորից ենթավեկտոր ստանալու 4 ձև՝ 1. դրական ամբողջ թվերի միջոցով, 2. բացասական ամբողջ թվերի միջոցով, 3. փրամաբանական արժեքների միջոցով, և 4. անունների միջոցով:

- Պարզագույն ձևը դրական ամբողջ թվերի միջոցով ենթավեկտոր ստանալն է:

Եթե փրված է որոշակի արժեքներից կազմված վեկտոր, ապա գրնելով այդ վեկտորի երկարությունը *length()* ֆունկցիայի միջոցով հնարավոր է 1-ից (հաշվումը **R**-ում սկսվում է 1-ից) մինչ երկարությունն ընկած թվերի միջոցով ստանալ վեկտորի համապարասխան կոորդինատը կամ կոորդինատները՝ օգտագործելով [ ] փակագծերը՝ վեկտորի անունից հետո

```
x<-c("a", "b", "c", "d", "e", "f", "g", "h", "i")
x[1]
```

```
## [1] "a"
```

```
x[5]
```

```
## [1] "e"
```

Նհարավոր է նաև միաժամանակ ստանալ համասեռ վեկտորի մի քանի կոորդինատներ, այդպիսով ստանալով փրված վեկտորի ենթավեկտոր՝

```
x[1:5]
```

```
## [1] "a" "b" "c" "d" "e"
```

```
x[c(1,3)]
```

```
## [1] "a" "c"
```

```
x[c(length(x)-1,length(x))] #the last two elements
```

```
## [1] "h" "i"
```

```
y<-c(1:3,5)
```

```
x[y]
```

```
## [1] "a" "b" "c" "e"
```

```
x[]
```

```
## [1] "a" "b" "c" "d" "e" "f" "g" "h" "i"
```

Երբ ենթավեկտոր կառուցելիս օգտագործվում է սկզբնական վեկտորի երկարությունը գերազանցող թիվ, ապա ծրագրային սխալ փեղի չի ունենում, ոչ էլ զգուշացում է հայտնվում: Պարզապես վերադարձվում է անհայտ արժեք:

```
x<-c("1", "B", "Complex")
```

```
length(x)
```

```
## [1] 3
```

```
x[4]
```

```
## [1] NA
```

Այս առանձնահատկության շնորհիվ հնարավոր է նոր կոորդինատներ ավելացնել վեկտորին՝

```
x<-1:10
x[length(x)+1]<-0
x
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 0
```

```
x[length(x)+10]<--1
x
```

```
## [1] 1 2 3 4 5 6 7 8 9 10 0 NA NA NA NA NA NA NA NA -1
```

- Ենթավեկոր սրանալու հաջորդ ձևը բացասական ամբողջ թվեր օգտագործելն է: Եթե դրական ամբողջ թվերն օգտագործվում են ընդհանուր համար անհրաժեշտ կոորդինատները, ապա բացասական ամբողջ թվերն օգտագործվում են դուրս թողնելու համար համապատասխան համարն ունեցող կոորդինատները: (Դրական և բացասական թվերի միաժամանակյա օգտագործումն արգելված է:)

```
x<-1:10
x[-c(1,2,3)]
```

```
## [1] 4 5 6 7 8 9 10
```

```
x[-1:-3]
```

```
## [1] 4 5 6 7 8 9 10
```

```
y<-seq(0,1,by=0.01)
y[x]
```

```
## [1] 0.00 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09
```

- Ենթավեկոր սրանալու հաջորդ ձևը փրամաբանական արժեքներ պարունակող, սկզբնական վեկորի երկարությանը հավասար երկարություն ունեցող վեկոր օգտագործելն է, որը *TRUE* արժեքների միջոցով ցույց կրա թե սկզբնական վեկորի որ կոորդինատներն է պետք ընտրել:

```
x<-1:4
y<-c(T,F,T,F)
x[y]
```

```
## [1] 1 3
```

Ենթավեկոր ընտրելու այս ձևն առավելապես օգտակար է, երբ փրամաբանական արժեքներ պարունակող վեկորը սրացվել է սկզբնական վեկորի հետ համեմատման գործողություններ կատարելիս: Օրինակ, ենթադրենք համասեռ, թվային վեկորից պետք է հեռացնել բոլոր բացասական թվերը, ապա նախ կստեղծենք փրամաբանական արժեքներ պարունակող վեկոր, որը *FALSE* արժեքներ կպարունակի բացասական արժեքներին համապատասխան, այնուհետև, օգտագործելով այդ վեկորը հնարավոր է սկզբնական վեկորից համապատասխան ենթավեկոր սրանալ:

```
x<--10:10
x
```

```
## [1] -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6
## [18] 7 8 9 10
```

```
x[x>=0]
```

```
## [1] 0 1 2 3 4 5 6 7 8 9 10
```

Այս հնարքը հաճախ օգտագործվում է վեկորից անհայտ արժեքները հեռացնելու համար՝

```
x<-c(10,NA,-5,NA,NA,0,-33,12.4)
x[!is.na(x)]
```

```
## [1] 10.0 -5.0 0.0 -33.0 12.4
```

```
x[!is.na(x) & x>=0]
```

```
## [1] 10.0 0.0 12.4
```

Այս դեպքում նույնպես եթե ենթավեկտոր սրանալու համար նախատեսված փրամաբանական վեկտորը կարճ է սկզբանական վեկտորից, ապա այն կրկնվում է մինչ սկզբնական վեկտորի երկարությանը հասնելը՝

```
x<-1:3
```

```
x[c(TRUE,FALSE)] #the same as x[c(TRUE,FALSE,TRUE)]
```

```
## [1] 1 3
```

- Ենթավեկտոր սրանալու վերջին ձևը անունների կոորդինատներին փրված անունների միջոցով է: Նախ քննարկենք, թե ինչպես է հնարավոր համասեռ վեկտորի կոորդինատներին անուններ փոխել:

```
x<-c(a=1,b=2/3,c=sqrt(3),number=-12)
```

```
x
```

```
##          a          b          c    number
## 1.0000000 0.6666667 1.7320508 -12.0000000
```

Ուշադրություն դարձրեք, որ այսօրեղ օգտագործվում է “=” նշանը:

Անունները հնարավոր է սրանալ `names()` ֆունկցիայի միջոցով՝

```
names(x)
```

```
## [1] "a"      "b"      "c"      "number"
```

Իրականում վեկտորին հնարավոր է նաև անուններ փոխել այս ֆունկցիայի միջոցով, կամ փոխել նախկինում փրված անունները՝

```
x<-c(a=1,b=2/3,c=sqrt(3),number=-12)
```

```
x
```

```
##          a          b          c    number
## 1.0000000 0.6666667 1.7320508 -12.0000000
```

```
names(x) <- c("Integer", "Rational", "Irrational", "Negative")
```

```
x
```

```
## Integer Rational Irrational Negative
## 1.0000000 0.6666667 1.7320508 -12.0000000
```

Եթե վեկտորի կոորդինատներին վերագրում ենք անունների վեկտոր, որն ավելի կարճ է, ապա կարճ վեկտորի կրկնում փրեղի ՉԻ ունենում, այլ պարզապես վերջին կոորդինատներն անուններ չեն սրանում՝

```
y<-x
```

```
y
```

```
## Integer Rational Irrational Negative
## 1.0000000 0.6666667 1.7320508 -12.0000000
```

```
names(y)<-"A name"
```

```
y
```

```
## A name <NA> <NA> <NA>
## 1.0000000 0.6666667 1.7320508 -12.0000000
```

Եթե համասեռ վեկտորի կոորդինատներն ունեն անուններ, ապա ենթավեկտոր կարելի է սրանալ նաև այսպես՝

```
x
```

```
## Integer Rational Irrational Negative
```

```
## 1.0000000 0.6666667 1.7320508 -12.0000000
```

```
x["Rational"]
```

```
## Rational
```

```
## 0.6666667
```

```
x[c("Rational","Irrational")]
```

```
## Rational Irrational
```

```
## 0.6666667 1.7320508
```

Անունով ենթավեկորը ընտրելն ավելի նախընտրելի է քանի որ համասեռ վեկորների հետ գործողություններ կատարելիս և դրանց երկարությունը փոխելիս կոորդինատի նախնական դիրքը կարող է փոխվել, իսկ անունը կմնա նույնը: Նաճախ հնարավոր է նախ կոորդինատների դիրքերի փոփոխություն՝ գոյություն ունեցող վեկորի ներսում:

```
# extract the irrational number
```

```
x[3]
```

```
## Irrational
```

```
## 1.732051
```

```
x["Irrational"]
```

```
## Irrational
```

```
## 1.732051
```

```
#extract the irrational number after adding the 0 number
```

```
y<-c(0,x);y
```

```
## Integer Rational Irrational Negative
```

```
## 0.0000000 1.0000000 0.6666667 1.7320508 -12.0000000
```

```
y[3]
```

```
## Rational
```

```
## 0.6666667
```

```
y["Irrational"]
```

```
## Irrational
```

```
## 1.732051
```

Կոորդինատի անվանման միջոցով կոորդինատը ուղիղ ձևով հեռացնել հնարավոր չէ, բայց հնարավոր է անվան միջոցով կոորդինատի համարը գրնել և դրան միջոցով այն օգտագործել: Օգտագործելու ենք *which()* ֆունկցիան, որը ստանալով փրամաբանական, համասեռ վեկոր վերադարձնում է այն համարները որտեղ *TRUE* է գրված՝

```
z<-c(T,F,F,T,F,F,T)
```

```
which(z)
```

```
## [1] 1 4 7
```

Ներկայումս, իռացիոնալ թիվը վերը սահմանված վեկորից հեռացնելու համար նախ պետք է գրնել այդ կոորդինատի համարը՝

```
x
```

```
## Integer Rational Irrational Negative
```

```
## 1.0000000 0.6666667 1.7320508 -12.0000000
```

```
names(x)=="Irrational"
```



```
## [1] FALSE FALSE TRUE FALSE
```

```
x[~which(names(x)=="Irrational")]
```

```
## Integer Rational Negative
## 1.0000000 0.6666667 -12.0000000
```

Իհարկե, հնարավոր է նաև փրկված անունով կոորդինատը հեռացնել ընտրելով միայն այն կոորդինատները, որոնց անունը հավասար չէ փրկված անվանը:

```
x[names(x)!="Irrational"]
```

```
## Integer Rational Negative
## 1.0000000 0.6666667 -12.0000000
```

Այս երկու մեթոդներով կարող ենք հեռացնել նաև մեկից ավելի կոորդինատներ՝ իրենց անունների միջոցով:

```
x[~which(names(x)%in%c("Irrational", "Rational"))]
```

```
## Integer Negative
## 1 -12
```

```
x[!names(x)%in%c("Irrational", "Rational")]
```

```
## Integer Negative
## 1 -12
```

Էրատոսթենեսի մաղը (The Sieve of Eratosthenes )

Տրված բնական թվերի վերջավոր հաջորդականություն և պետք է հեռացնել դրանցից թվերն այնպես, որ մնան միայն պարզ թվերը: Ալգորիթմը հետևյալն է՝ նախ հեռացվում է 1 թիվը, այնուհետև հեռացվում են 2-ի վրա բաժանվող թվերը, բայց ոչ 2-ն ինքը և այսպես շարունակ հեռացվում են հաջորդ բնական թվի վրա բաժանվող թվերը: Այս գործընթացը կարիք չկա շարունակելու մինչ սկզբում փրկված ամենամեծ բնական թվի արմատը չզերազանցող բնական թիվը՝

```
start<-Sys.time()
n<-1000
X<-seq_len(n)
X<-X[X>1]
for(x in X){
  X<-X[X%%x!=0 | X==x]
}
end<-Sys.time()
X
```

```
## [1] 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59
## [18] 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139
## [35] 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233
## [52] 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337
## [69] 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439
## [86] 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557
## [103] 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653
## [120] 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757 761 769
## [137] 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883
## [154] 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997
```

```
end-start
```

```
## Time difference of 0.01300502 secs
```

*for()* կառուցվածքին մանրամասնորեն դեռևս կանդորադառնանք, իսկ *Sys.time()* ֆունկցիան վերադարձնում է ժամանակի այն պահը երբ այն կանչվել է:

Կանչելով այդ ֆունկցիան մինչ ծրագրի սկիզբը և ամենավերջում, ապա վերցնելով ժամանակի այդ պահերի փարբերությունը կստանանք թե ինչքան ժամանակ է պահանջվել ծրագրից հաշվարկները կատարելու համար: Այդպես հնարավոր է ստուգել գրված ծրագրի արդյունավետությունը՝ կախված նրանից թե ինչքան արագ է այն կատարվում:

### 2.2.7 Օբյեկտների հատկանիշեր (Attributes)

Նաճախ **R**-ում օբյեկտներն ունեն հատկանիշեր (attributes), որոնցից են

- names, dimnames - անուններ, չափողականության անուններ
- dimensions չափողականություններ
- class - տեսակ
- օգտագործողի կողմից սահմանված այլ հատկանիշերը

Օբյեկտի բոլոր հատկանիշերը կարելի է ստանալ `attributes()` ֆունկցիայի միջոցով: Օրինակ հնարավոր է համասեռ վեկտորի արժեքներին անուններ փայլ՝

```
x<-c(a=1,b=12)
x
```

```
## a b
## 1 12
```

```
class(x)
```

```
## [1] "numeric"
```

```
attributes(x)
```

```
## $names
## [1] "a" "b"
```

Ուշադրություն դարձրեք, որ այսպիսի օգտագործվում է “=” նշանը:

`names` հատկանիշը հնարավոր է ստանալ `attr()` ֆունկցիայի միջոցով՝

```
attr(x,"names")
```

```
## [1] "a" "b"
```

Նիմանական հատկանիշները, ինչպիսիք են չափողականությունը, տեսակը, անունները և չափողականությունների անունները հնարավոր է ստանալ նաև համապատասխան `names()`, `dim()`, `class()`, `dimnames()` ֆունկցիաների միջոցով՝

```
names(x) #attr(x,"names")
```

```
## [1] "a" "b"
```

Օգտագործողը հնարավորություն ունի սահմանելու նոր հատկանիշեր: Ինչպես նշվեց **R**-ում փոփոխականի անունները սահմանելիս տեղի ունեն որոշակի սահմանափակումներ: Այդ պատճառով, եթե օգտագործողն ուզում է փոփոխականի վերաբերյալ լրացուցիչ փոխալ պահել, ապա կարող է սահմանել պիտակ `label`, և այդպիսի պահել փոփոխականի վերաբերյալ ավելի լիակատար նկարագրություն՝

```
years<-2012:2015
names(years)<-c("Y1","Y2","Y3","Y4")
attr(years,"label")<-"Years of data collection"
years
```

```
## Y1 Y2 Y3 Y4
## 2012 2013 2014 2015
```

```
## attr("label")
## [1] "Years of data collection"
```

```
attributes(years)
```

```
## $names
## [1] "Y1" "Y2" "Y3" "Y4"
##
## $label
## [1] "Years of data collection"
```

Վեկտորը փոփոխության ենթարկելիս դրա հարկանիշները կորում են, բացառությամբ վերը նշված հիմնական հարկանիշների՝

```
years<-years[-1]
years
```

```
##   Y2   Y3   Y4
## 2013 2014 2015
```

```
attributes(years)
```

```
## $names
## [1] "Y2" "Y3" "Y4"
```

### 2.2.8 Մատրիցներ (Matrices)

Վեկտորների միջոցով կարող ենք սրանալ այլ կառուցվածքներ, մասնավորապես մատրիցներ, որոնք վեկտորներ են լրացուցիչ՝ չափողականություն հարկանիշով: Չափողականությունն իր հերթին համասեռ, ամբողջ թվերից կազմված վեկտոր է՝ երկու երկարությամբ և իր մեջ պարունակում է փողերի և սյուների քանակը: Վերցնենք համասեռ, թվային վեկտոր և դրան հաղորդենք չափողականություն հարկանիշը և տեսնենք ինչ է սրացվում՝

```
x<-1:10
x
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
attributes(x)
```

```
## NULL
```

Նամասեռ վեկտորը հարկանիշներ չունի: Այս վեկտորի համար սահմանենք չափողականություն՝

```
dim(x)<-c(2,5)
attributes(x)
```

```
## $dim
## [1] 2 5
```

```
dim(x)
```

```
## [1] 2 5
```

Այժմ տեսնենք թե ինչպես է փոխվել համասեռ վեկտորի տեսքը և ինչպիսին է դրա նոր կառուցվածքը՝

```
x
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

```
class(x)
```

```
## [1] "matrix"
```

Մափրից ստեղծելու պարզագույն ձևը *matrix()* ֆունկցիայի միջոցով է՝

```
x<-matrix()
```

```
attributes(x)
```

```
## $dim
```

```
## [1] 1 1
```

```
x
```

```
##      [,1]
```

```
## [1,]  NA
```

```
x<-matrix(nrow=2,ncol=3)
```

```
x
```

```
##      [,1] [,2] [,3]
```

```
## [1,]  NA  NA  NA
```

```
## [2,]  NA  NA  NA
```

Ստեղծեցինք դափարկ մափրից, որը ծրագրի կողմից լրացվեց *NA* հատուկ արժեքներով: Ինչպես տեսնում ենք՝ ցանկացած մափրից ստեղծելիս ստեղծվում է նաև դրա չափողականություն հատկանիշը: Օգտագործելով *dim()* ֆունկցիան՝ մենք կարող ենք ստանալ մափրիցի չափողականությունը պարունակող վեկտորը՝

```
attributes(x)
```

```
## $dim
```

```
## [1] 2 3
```

```
dim(x)
```

```
## [1] 2 3
```

Այսպես առաջին թիվը ցույց է տալիս տողերի քանակը, իսկ երկրորդը՝ սյուների: Կարող ենք նաև օգտագործել *nrow()*, *ncol()* ֆունկցիաները ստանալու համար համապատասխանաբար մափրիցի տողերի և սյուների քանակները:

```
nrow(x);ncol(x)
```

```
## [1] 2
```

```
## [1] 3
```

Մափրիցը ստեղծելիս մենք կարող էինք տալ այն արժեքները որոնցով կառուցվում է մափրիցը՝ համասեռ վեկտորի տեսքով՝

```
x<-matrix(data=1:6, ncol=3, nrow=2)
```

```
x
```

```
##      [,1] [,2] [,3]
```

```
## [1,]   1   3   5
```

```
## [2,]   2   4   6
```

Մափրիցները կազմվում են սյուններով, այսինքն՝ փրված համասեռ վեկտորի արժեքները տեղադրվում են վերևի ձախ անկյունից և ներքև իջնում սյուններով: Տող առ տող լրացնելու համար պետք է արգումենտներից մեկում փոխել լռելյալ արժեքը (default value)

```
x<-matrix(data=1:6, ncol=3, nrow=2, byrow = TRUE)
```

```
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

Մատրիցներ ստեղծելու վերջին ձևը համասեռ վեկտորները իրար որպես սյուններ կամ տողեր միացնելն է, որոնք կապարվում են `cbind()`, `rbind()` ֆունկցիաների միջոցով`

```
x<-12:14
y<-28:30
m<-rbind(x,y)
m
```

```
##      [,1] [,2] [,3]
## x    12   13   14
## y    28   29   30
```

```
class(m)
```

```
## [1] "matrix"
```

```
m<-cbind(x,y)
m
```

```
##      x y
## [1,] 12 28
## [2,] 13 29
## [3,] 14 30
```

Նիշենք, որ մատրիցը կարող է պարունակել միայն նույն տեսակի արժեքներ, բայց ոչ պարպաղիթ թվային`

```
x<-matrix(data=c("a","b","c","d"),ncol=2,nrow=2)
class(x)
```

```
## [1] "matrix"
```

```
x
```

```
##      [,1] [,2]
## [1,] "a"  "c"
## [2,] "b"  "d"
```

```
typeof(x)
```

```
## [1] "character"
```

Ինչպես և համասեռ վեկտորների դեպքում, եթե փորձենք մատրիցին ավելացնել այլ տեսակի սյուն կամ տող, տրեղի կունենա տեսակի քողարկված փոփոխություն`

```
x<-matrix(data=1:6, ncol=3, nrow=2, byrow = TRUE)
typeof(x)
```

```
## [1] "integer"
```

```
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
x_<-rbind(x,c("a","b","c"))
typeof(x_)
```

```
## [1] "character"
```

```
x_
```

```
##      [,1] [,2] [,3]
## [1,] "1"  "2"  "3"
## [2,] "4"  "5"  "6"
## [3,] "a"  "b"  "c"
```

### 2.2.9 Գործողություններ մափրիցների հետ

Թվային մափրիցների հետ գործողությունների կանոնները գրեթե նույնն են ինչ համասեռ, թվային վեկտորների հետ գործողությունների կանոնները: Նախ կիրառենք `length()` ֆունկցիան որևէ մափրիցի վրա՝

```
x<-matrix(data=1:6, ncol=3, nrow=2, byrow = TRUE)
length(x)
```

```
## [1] 6
```

Այն վերադարձնում է մափրիցի էլեմենտների քանակը: Այս մափրիցի հետ թվաբանական գործողություններ կատարելիս [R](#)-ը դրան վերաբերվում է որպես սովորական համասեռ, թվային վեկտորի՝ կարողացված սյուն առ սյուն:

```
y<-1:6
x+y
```

```
##      [,1] [,2] [,3]
## [1,]  2   5   8
## [2,]  6   9  12
```

```
x*y
```

```
##      [,1] [,2] [,3]
## [1,]  1   6  15
## [2,]  8  20  36
```

```
x/y
```

```
##      [,1]      [,2] [,3]
## [1,]  1 0.6666667 0.6
## [2,]  2 1.2500000 1.0
```

```
x^y
```

```
##      [,1] [,2] [,3]
## [1,]  1   8  243
## [2,] 16 625 46656
```

Նույն արդյունքը կստացվեր, եթե `y` համասեռ վեկտորը մափրից լիներ՝

```
y<-matrix(y,ncol=3)
x+y
```

```
##      [,1] [,2] [,3]
## [1,]  2   5   8
## [2,]  6   9  12
```

```
x*y
```

```
##      [,1] [,2] [,3]
## [1,]  1   6  15
## [2,]  8  20  36
```

```
x/y
```

```
##      [,1]      [,2] [,3]
## [1,]    1 0.6666667 0.6
## [2,]    2 1.2500000 1.0
```

```
x^y
```

```
##      [,1] [,2] [,3]
## [1,]    1    8 243
## [2,]   16 625 46656
```

Նմանապես նույնն են նաև փոխված մափրիցի և ավելի կարճ երկարություն ունեցող վեկտորի (կամ մափրիցի) հետ գործողություններ կատարելու կանոնները՝

```
x
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
x+1
```

```
##      [,1] [,2] [,3]
## [1,]    2    3    4
## [2,]    5    6    7
```

```
x+c(1,2)
```

```
##      [,1] [,2] [,3]
## [1,]    2    3    4
## [2,]    6    7    8
```

```
x+c(1,2,3)
```

```
##      [,1] [,2] [,3]
## [1,]    2    5    5
## [2,]    6    6    9
```

```
x+c(1,2,3,4)
```

```
## Warning in x + c(1, 2, 3, 4): longer object length is not a multiple of
## shorter object length
```

```
##      [,1] [,2] [,3]
## [1,]    2    5    4
## [2,]    6    9    8
```

Կամ փրամաբանական գործողություններ կատարելիս՝

```
x>c(1,2,3)
```

```
##      [,1] [,2] [,3]
## [1,] FALSE FALSE TRUE
## [2,]  TRUE  TRUE  TRUE
```

Պետք է ուշադիր լինել, որ գումարվող մափրիցը նույն չափողականությունն ունենա ինչ սկզբնական մափրիցը կամ առհասարակ չափողականություն չունենա՝

```
x+matrix(1)
```

```
## [1] "Error in x + matrix(1) : non-conformable arrays\n"
```

Բնականաբար կառուցվածքի մաթրից լինելու հանգամանքը նաև լրացուցիչ հնարավորություններ է ընձեռում: Օրինակ հնարավոր է մաթրիցային բազմապարկում կատարելը`

```
x<-matrix(1:4,ncol=2)
```

```
x
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
y<-matrix(c(1,0,0,1),ncol=2)
```

```
y
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

```
x%*%x
```

```
##      [,1] [,2]
## [1,]    7   15
## [2,]   10   22
```

```
x%*%y
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

Քառակուսային, չվերաստրված (ոչ գրոյական որոշիչ ունեցող) մաթրիցի հակադարձը հնարավոր է հաշվել `solve()` ֆունկցիայի միջոցով: Քառակուսային մաթրիցի որոշիչը հնարավոր է հաշվել `det()` ֆունկցիայի միջոցով`

```
x
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
det(x)
```

```
## [1] -2
```

```
solve(x)
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

```
x%*%solve(x)
```

```
##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1
```

Մաթրիցային հանրահաշվի կարևորագույն գործողություններից է մաթրիցի փրասնպրնացումը` գլխավոր անկյունագծի նկատմամբ շրջումը`

```
x
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```



```
t(x)

##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4

x<-matrix(letters[1:6],ncol=3)
x
```

```
##      [,1] [,2] [,3]
## [1,] "a"  "c"  "e"
## [2,] "b"  "d"  "f"
```

```
t(x)

##      [,1] [,2]
## [1,] "a"  "b"
## [2,] "c"  "d"
## [3,] "e"  "f"
```

### 2.2.10 Ենթամատրից (Subsetting Matrices)

Մատրիցից ենթամատրից ստանալիս նույնպես կարող ենք վարվել ինչպես ենթավեկտոր ստանալիս՝

```
x<-matrix(1:24, ncol=6)
x

##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    5    9   13   17   21
## [2,]    2    6   10   14   18   22
## [3,]    3    7   11   15   19   23
## [4,]    4    8   12   16   20   24
```

```
x[1]
```

```
## [1] 1
```

```
x[3]
```

```
## [1] 3
```

Բայց ավելի հարմար է մատրիցի էլեմենտները հանել կրկնակի՝ փողի և սյան համարներով: Առաջին համարը ցույց է տալիս փողը որտեղ գտնվում է հետաքրքրող էլեմենտը, իսկ երկրորդը՝ սյունը:

```
x

##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    5    9   13   17   21
## [2,]    2    6   10   14   18   22
## [3,]    3    7   11   15   19   23
## [4,]    4    8   12   16   20   24
```

```
x[1,2]
```

```
## [1] 5
```

```
x[1:2,c(3,5,6)]
```

```
##      [,1] [,2] [,3]
## [1,]    9   17   21
## [2,]   10   18   22
```

Վերջին գործողության ժամանակ մենք նշել ենք երկու փողի և երեք սյուների համարներ: Վերադարձվել են  $2 \times 3$  էլեմենտներ՝ այդ երկու փողի և երեք սյան հարման կետերում հայտնված էլեմենտներից կազմված ենթամատրիցը՝

Figure 2.1: Ենթամատրից

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	1	5	9	13	17	21
[2,]	2	6	10	14	18	22
[3,]	3	7	11	15	19	23
[4,]	4	8	12	16	20	24

Նախավոր է նաև ենթամատրից ստանալիս վերադարձնել բոլոր փողերը կամ բոլոր սյուները՝ դափարկ թողնելով համապատասխան համարի տեղը՝

```
x[,c(1,4)]
```

```
##      [,1] [,2]
## [1,]   1  13
## [2,]   2  14
## [3,]   3  15
## [4,]   4  16
```

```
x[c(3,2),]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]   3   7  11  15  19  23
## [2,]   2   6  10  14  18  22
```

```
x[c(2,3,2),]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]   2   6  10  14  18  22
## [2,]   3   7  11  15  19  23
## [3,]   2   6  10  14  18  22
```

Ուշադրություն դարձրեք, որ հնարավոր է փողերը կամ սյուները ընկրել մատրիցում իրենց հերթականությանը հակառակ, կամ էլ՝ որոշ փողեր և սյուներ մի քանի անգամ:

Դափարկ թողնելով միաժամանակ և՛ սյուների համարների տեղը, և՛ փողերի կվերադարձնենք ամբողջ մատրիցը: Սա նույնն է թե ընդհանրապես ենթամատրիցի գործողությունը չկիրառելը, սակայն այս գործողությունը թույլ է տալիս միաժամանակ փոխարինել մատրիցի բոլոր էլեմենտները՝

```
x
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]   1   5   9  13  17  21
## [2,]   2   6  10  14  18  22
## [3,]   3   7  11  15  19  23
## [4,]   4   8  12  16  20  24
```

```
x[,]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]   1   5   9  13  17  21
## [2,]   2   6  10  14  18  22
## [3,]   3   7  11  15  19  23
## [4,]   4   8  12  16  20  24
```

```
x[,]<-0 #x<-0 will delete the matrix and create a new variable equal to 0
x
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  0   0   0   0   0   0
## [2,]  0   0   0   0   0   0
## [3,]  0   0   0   0   0   0
## [4,]  0   0   0   0   0   0
```

Ի փոքրերություն համասեռ վեկտորների, որպեսզի վեկտորի երկարությունից մեծ համարով կոորդինատներ կանչելու դեպքում պարզապես անհայտ արժեքն էր վերադարձվում, երկու համարների միջոցով մաքրիցի էլեմենտ կանչելիս փողերի կամ սյուների քանակները գերազանցող համար կանչելու դեպքում փեղի կունենա սխալ՝

```
x<-matrix(1:4,ncol=2)
x
x[1,3]
```

```
##      [,1] [,2]
## [1,]  1   3
## [2,]  2   4
## [1] "Error in x[1, 3] : subscript out of bounds\n"
```

### 2.2.11 Ցուցակներ (Lists)

Նամասեռ վեկտորներից հետո վեկտորի կարևոր օրինակներից են ցուցակները: Մրանք փոխալների կառուցվածքների յուրահատուկ օրինակ են նրանով, որ կարող են պահել փոքրեր տեսակի արժեքներ, նաև փոքրեր տեսակի կառուցվածքներ, այդ թվում՝ այլ ցուցակներ:

```
x<-list(1,"a",TRUE,1+4i)
x
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+4i
```

```
class(x)
```

```
## [1] "list"
```

```
length(x)
```

```
## [1] 4
```

```
y<-list(1:10,x)
```

```
length(y)
```

```
## [1] 2
```

```

y
## [[1]]
## [1] 1 2 3 4 5 6 7 8 9 10
##
## [[2]]
## [[2]][[1]]
## [1] 1
##
## [[2]][[2]]
## [1] "a"
##
## [[2]][[3]]
## [1] TRUE
##
## [[2]][[4]]
## [1] 1+4i

```

Ինչպես տեսնում եք վերը նշված ցուցակի երկարությունը երկուս է (թեև երկրորդ կոորդինատն ինքը ցուցակ է): Ցուցակներում, ինչպես նաև համասեռ վեկտորներում, հնարավոր է դրանում առկա արժեքներին անուններ փալ

```

x<-list(b=1,a="a",c=TRUE,e=1+4i)
x

```

```

## $b
## [1] 1
##
## $a
## [1] "a"
##
## $c
## [1] TRUE
##
## $e
## [1] 1+4i

```

Այս դեպքում դիտելով այս ցուցակի հատկանիշները մենք նկատում ենք, որ անունները (*names*) հայտնվել է հատկանիշների մեջ

```
attributes(x)
```

```

## $names
## [1] "b" "a" "c" "e"

```

Սա նշանակում է, որ *names()* ֆունկցիան կարող ենք կիրառել մեր ստեղծած ցուցակի վրա և ստանալ նրանում պահվող արժեքների անունները՝

```
names(x)
```

```
## [1] "b" "a" "c" "e"
```

### 2.2.12 Ֆակտորներ (Factors)

Ֆակտորները ևս վեկտորների տարատեսակ են: Դրանք նախատեսված են կարեգործիաների վերլուծություն կատարելու համար՝ վիճակագրական մոդելավորման մեջ: Ֆակտորում իրարից տարբեր արժեքները առանձին պահվում են որպես հատկանիշ և կոչվում են պիտակ՝ (*label*): Ֆակտորները լինում են կարգավորված և ոչ

կարգավորված: Ֆակտորի ամենապարզ օրինակը է 0 և 1-երից կազմված վեկտոր որտեղ 0-ն ունենա Female (կին), իսկ 1-ը՝ Male (տղամարդ) պիտակը:

Ֆակտորները կարող են ստեղծվել `factor()` ֆունկցիայի միջոցով, փառվ նրան որպես արգումենտ համասեռ վեկտոր՝ տեքստային տեսակի,

```
x<-factor(c("yes", "yes", "no", "yes", "yes", "no"))
class(x)
```

```
## [1] "factor"
```

```
x
```

```
## [1] yes yes no yes yes no
```

```
## Levels: no yes
```

```
attributes(x)
```

```
## $levels
```

```
## [1] "no" "yes"
```

```
##
```

```
## $class
```

```
## [1] "factor"
```

```
levels(x)
```

```
## [1] "no" "yes"
```

Ինչպես տեսնում եք վերևում՝ պիտակների կարգը որոշվում է այբբենական դասավորությամբ: Եթե կարիք կա փոխել ֆակտորում պիտակների դասավորման կարգը, ապա պիտակները կարելի է նախապես հայտարարել՝ ֆակտորը ստեղծելիս

```
x<-factor(c("yes", "yes", "no", "yes", "yes", "no"), levels=c("yes", "no"))
```

```
x
```

```
## [1] yes yes no yes yes no
```

```
## Levels: yes no
```

Ֆակտորները կարող են հեշտացնել փոխալային հետ աշխարհում: Օրինակ, եթե փրված է մեծ վեկտոր, որում գրված են 0, 1 արժեքները և պետք է կատարել արժեքների փոփոխություն՝ 0 = FEMALE, 1 = MALE,

```
x<-c(rep(0,10), rep(1,10))
```

```
x
```

```
## [1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
```

```
levels(x)
```

```
## NULL
```

```
levels(x)<-c("Male", "Female")
```

```
x
```

```
## [1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
```

```
## attr(,"levels")
```

```
## [1] "Male" "Female"
```

Ֆակտորների հետ թվաբանական գործողություններ հնարավոր չէ անել անգամ երբ դրանում թվային արժեքներ են պահված: Ֆակտորը թվային վեկտորի վերածելիս պետք է ուշադրություն դարձնել հետևյալ կարևոր առանձնահատկությանը: Դիտարկենք հետևյալ օրինակը՝

```
x<-rep(c(0,4), times=10)
```

```
x<-factor(x)
```

```
x
```

```
## [1] 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4
## Levels: 0 4
```

Փորձենք նշված ֆակտորը վերածել թվային, համասեռ վեկտորի՝

```
as.numeric(x)
```

```
## [1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
```

Պարզապես տեսնելով է ունեցել պիտակների համարակալում և արժեքների փոխարինում իրենց պիտակներով: Թվային արժեքները պահպանելու համար պետք է հետևյալ կերպ վարվել՝

```
as.numeric(as.character(x))
```

```
## [1] 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4 0 4
```

Պիտակները նաև օգտակար են այն դեպքերում երբ հնարավոր արժեքները գիտենք, բայց այդ արժեքներից որոշները փոխարինում առկա չեն: Օրինակ, սեռը երկու հնարավոր արժեք ունի, բայց հնարավոր է որոշ դեպքում հավաքագրված փոխարինում միայն արական սեռը ցույց տալով արժեքներ: Այս դեպքում նպատակահարմար է բոլոր հնարավոր արժեքները պահել որպես պիտակներ՝

```
x<-factor(c("M", "M", "M", "M"), levels=c("M", "F"))
x
```

```
## [1] M M M M
## Levels: M F
```

Օրինակ `table()` ֆունկցիան հաշվում է իրարից փոքր արժեքների կրկնումների քանակները վեկտորում: Ներկայացրեք, երբ հնարավոր արժեքներից մեկը ներկայացված չէ փոխարինում, ապա միևնույնն է նպատակահարմար է այն ընդգրկել վերջնական հաշվարկում 0 քանակով՝

```
table(c("M", "M", "M", "M"))
```

```
##
## M
## 4
```

```
table(x)
```

```
## x
## M F
## 4 0
```

### 2.2.13 Ուղղանկյունաձև փոխարկումները (Data Frames)

Տվյալների պահպանման հաջորդ կառուցվածքը ուղղանկյունաձև կառուցվածքն է՝ Data Frame, որի մեջ պահվում են աղյուսակային փոխարկումները: Այն ստեղծելու համար օգտագործվում է `data.frame()` ֆունկցիան: Այս կառուցվածքը շատ նման է մատրիցային, մեկ կարևոր փոքրությամբ, որ եթե մատրիցում բոլոր փոքրերը պետք է նույն տեսակի ունենային, ապա ուղղանկյունաձև կառուցվածքում յուրաքանչյուր սյուն կարող է փոքրեր տեսակի լինել, բայց սյունի ներսում պետք է նույն տեսակի արժեքները պահպանվեն: Բնականաբար նաև սյուները պետք է միևնույն երկարություն ունենան: Այլ կերպ ասած՝ ուղղանկյունաձև կառուցվածքում սյուները միևնույն երկարություն ունեցող համասեռ վեկտորներ են՝ ընդհանուր դեպքում փոքրեր տեսակների:

Ուղղանկյունաձև կառուցվածքն ունի հարկադիր `row.names`, որը փոքրերի անուններն են և կարող են ստացվել `row.names()` ֆունկցիայի կանչմամբ, ինչպես նաև սյունների անուններ, որոնք կարող են ստանալ `names()` ֆունկցիայի միջոցով:

Պիտակները հետևյալ օրինակը, որում ստեղծվում է փոխարկում ուղղանկյունաձև կառուցվածք՝

```
d<-data.frame(subjects=1:3,status=c(TRUE,TRUE,FALSE))
d
```

```
##  subjects status
##  1          1  TRUE
##  2          2  TRUE
##  3          3 FALSE
```

```
attributes(d)
```

```
## $names
## [1] "subjects" "status"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3
```

```
nrow(d)
```

```
## [1] 3
```

```
ncol(d)
```

```
## [1] 2
```

```
dim(d)
```

```
## [1] 3 2
```

Ընդգծենք, կարևոր առանձնահատկություն՝ հնարավոր է անուններ փոխել նաև մատրիցների սյուներին և փողերին, բայց փողերի և մատրիցների անունները չեն հանդիսանում մատրիցային կառուցվածքի հատկանիշ, այն է՝ դրանք չեն ստեղծվում կառուցվածքի ստեղծման հետ մեկտեղ (մատրիցի դեպքում դրանք կոչվում են *dimnames* հատկանիշ, այսինքն՝ չափողականության անուններ):

```
m<-matrix(1:12,ncol=3,byrow=TRUE)
row.names(m)<-nrow(m):1
colnames(m)<-c("a","b","c")
attributes(m)
```

```
## $dim
## [1] 4 3
##
## $dimnames
## $dimnames[[1]]
## [1] "4" "3" "2" "1"
##
## $dimnames[[2]]
## [1] "a" "b" "c"
```

```
m
```

```
##  a  b  c
## 4  1  2  3
## 3  4  5  6
## 2  7  8  9
## 1 10 11 12
```

Իսկ ուղղակյունաձև կառուցվածքի ստեղծման հետ մեկտեղ ստեղծվում են դրա փողերի և սյուների անուններ

հարկանիշները

```
d<-data.frame()
d
```

```
## data frame with 0 columns and 0 rows
```

```
attributes(d)
```

```
## $names
## character(0)
##
## $row.names
## integer(0)
##
## $class
## [1] "data.frame"
```

### 2.2.14 Ենթակառուցվածքներ (Subsetting)

Այսպեղ ամփոփելու ենք **R**-ի փվյալների կառուցվածքներից ենթակառուցվածքներ սրանալու հնարավորությունները: Ինչպես նշվեց վերևում՝ համասեռ վեկտորից հնարավոր է ենթավեկտոր սրանալ [ գործողության միջոցով՝ 4 ձևով: Իսկ ենթակառուցվածք սրանալու գործողությունները 3-ն են՝ [, [[, \$.

- [ գործողության կարևորագույն առանձնահատկությունն է, որ այն միշտ վերադարձնում է նույն տեսակի կառուցվածք ինչ սկզբնական կառուցվածքն է: Օրինակ, եթե դրա միջոցով ենթակառուցվածք ենք սրանում ցուցակից, ապա սրացված կառուցվածքը ևս ցուցակ է:

```
l<-list(a=data.frame(x=letters[1:4], y=seq(2,3,length.out=4)),
        b=list(pi,3.5,4),c=matrix(1:6,ncol=3))
length(l)
```

```
## [1] 3
```

```
l
```

```
## $a
##   x      y
## 1 a 2.000000
## 2 b 2.333333
## 3 c 2.666667
## 4 d 3.000000
##
## $b
## $b[[1]]
## [1] 3.141593
##
## $b[[2]]
## [1] 3.5
##
## $b[[3]]
## [1] 4
##
## $c
##      [,1] [,2] [,3]
## [1,]    1    3    5
```



```
## [2,] 2 4 6
```

Այս ցուցակի առաջին ենթաօբյեկտը ուղղանկյուն-փոխարկ է, բայց երբ փորձում ենք դրան դիմել [ գործողության միջոցով, ապա արդյունքը ստացվում է ցուցակ, որովհետև սկզբնական կառուցվածքը ցուցակ էր:

```
l[1]
```

```
## $a
## x y
## 1 a 2.000000
## 2 b 2.333333
## 3 c 2.666667
## 4 d 3.000000
```

```
class(l[1])
```

```
## [1] "list"
```

[ գործողության մյուս առանձնահատկությունն է, որ դրա միջոցով հնարավոր է դիմել կառուցվածքի մի քանի ենթաօբյեկտներին միաժամանակ՝

```
l[c(1,3)]
```

```
## $a
## x y
## 1 a 2.000000
## 2 b 2.333333
## 3 c 2.666667
## 4 d 3.000000
##
## $c
## [,1] [,2] [,3]
## [1,] 1 3 5
## [2,] 2 4 6
```

- Ենթակառուցվածք ստանալու հաջորդ գործողությունն է [[: Սրա միջոցով հնարավոր է դիմել կառուցվածքի միայն մեկ ենթաօբյեկտին և վերադարձվող ենթակառուցվածքի տեսակը կարող է չհամապատասխանել սկզբնական կառուցվածքի տեսակին՝

```
class(l[[1]])
```

```
## [1] "data.frame"
```

```
l[[1]]
```

```
## x y
## 1 a 2.000000
## 2 b 2.333333
## 3 c 2.666667
## 4 d 3.000000
```

- Ենթակառուցվածք ստանալու վերջին գործողությունը \$ է: Սա օգտագործվում է, երբ ուզում ենք կառուցվածքի ենթաօբյեկտներին դիմել անունների միջոցով: Այս գործողության դեպքում ևս հնարավոր է դիմել միայն մեկ ենթաօբյեկտի և վերադարձվող տեսակը կարող է չհամապատասխանել սկզբնական կառուցվածքի տեսակին:

```
class(l$a)
```

```
## [1] "data.frame"
```

```
l$a
```

```
##      x          y
## 1 a 2.000000
## 2 b 2.333333
## 3 c 2.666667
## 4 d 3.000000
```

Նախորդ երկու գործողությունների դեպքում ևս հնարավոր էր ենթակառուցվածք ստանալ օգտագործելով ենթաօբյեկտի անունը (դեռ ավելին՝ [-ի դեպքում հնարավոր էր նաև մի քանի անունների դիմել), բայց այս վերջինն իրենից ներկայացնում է ավելի կարճ գրելաձև:

```
l["a"]
```

```
## $a
##      x          y
## 1 a 2.000000
## 2 b 2.333333
## 3 c 2.666667
## 4 d 3.000000
```

```
l[["a"]]
```

```
##      x          y
## 1 a 2.000000
## 2 b 2.333333
## 3 c 2.666667
## 4 d 3.000000
```

```
l[c("b","c")]
```

```
## $b
## $b[[1]]
## [1] 3.141593
##
## $b[[2]]
## [1] 3.5
##
## $b[[3]]
## [1] 4
##
##
## $c
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

\$ գործողությունն ունի մեկ սահմանափակում՝ դրան անուն փոխանցելիս չենք կարող օգտվել փոփոխականներից, այսինքն՝

```
name<-"a"
```

```
l[name]
```

```
## $a
##      x          y
## 1 a 2.000000
## 2 b 2.333333
## 3 c 2.666667
```

```
## 4 d 3.000000
```

```
l[[name]]
```

```
##      x      y
## 1 a 2.000000
## 2 b 2.333333
## 3 c 2.666667
## 4 d 3.000000
```

```
l$name
```

```
## NULL
```

Առաջին երկուսը վերադարձնում են անհրաժեշտ ենթաօբյեկտը, բայց վերջինը՝ ոչ, քանի որ \$ գործողությանը պետք է փոխանցել անուն և ոչ թե անուն պարունակող փոփոխական:

Նշենք ևս մեկ կարևոր առանձնահատկության մասին: \$ նշանով ենթակառուցվածք սրանալիս կարելի է ենթաօբյեկտի անունը կիսապ գրել և փեղի կունենա մասնակի համապատասխանեցում՝ *partial matching* և կգտնվի համապատասխան անունով ենթաօբյեկտը՝

```
L<-list(asassdrfsas=1:10, b=15)
```

```
L$a
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
L[["a"]]
```

```
## NULL
```

```
L["a"]
```

```
## $<NA>
```

```
## NULL
```

[[ գործողության դեպքում կարելի է սրանալ նույն արդյունքը՝ համապատասխան արգումենտի արժեքը *FALSE*-ի փոխելով՝

```
L[["a",exact=FALSE]]
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

Ենթակառուցվածք սրանալու գործողությունները կարելի է կիրառել նաև հաջորդաբար:

```
l[[2]]
```

```
## [[1]]
## [1] 3.141593
##
## [[2]]
## [1] 3.5
##
## [[3]]
## [1] 4
```

```
l[[2]][[1]]
```

```
## [1] 3.141593
```

```
l$a
```

```
##      x      y
## 1 a 2.000000
## 2 b 2.333333
```

```
## 3 c 2.666667
## 4 d 3.000000
l$a$y
## [1] 2.000000 2.333333 2.666667 3.000000
l$a$y[3]
```

```
## [1] 2.666667
```

### 2.2.15 Ենթակառուցվածք ուղղանկյուն-փոխարկայինից (Subsetting a Data Frame)

Ուղղանկյուն-փոխարկայինի դեպքում ենթակառուցվածք սրահալու ձևերն ամենաբազմազանն են, քանի որ բոլոր նշված մեթոդները կիրառելի են:

`data.frame()` ֆունկցիան կարևոր առանձնահատկություն ունի՝ ուղղանկյուն-փոխարկային ստեղծելիս բոլոր տեսակի սյուները վերածվում են ֆակտորների: Այս գործողությունից խուսափելու համար պետք է ֆունկցիայի համապատասխան արգումենտին վերագրել `FALSE` արժեքը՝

```
df<-data.frame(x=letters[1:10],y=seq(pi,4,length.out=10))
df
```

```
##      x          y
## 1   a 3.141593
## 2   b 3.236971
## 3   c 3.332350
## 4   d 3.427728
## 5   e 3.523107
## 6   f 3.618486
## 7   g 3.713864
## 8   h 3.809243
## 9   i 3.904621
## 10  j 4.000000
```

```
df$x
```

```
## [1] a b c d e f g h i j
## Levels: a b c d e f g h i j
```

```
df<-data.frame(x=letters[1:10],y=seq(pi,4,length.out=10),stringsAsFactors = FALSE)
df$x
```

```
## [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
df
```

```
##      x          y
## 1   a 3.141593
## 2   b 3.236971
## 3   c 3.332350
## 4   d 3.427728
## 5   e 3.523107
## 6   f 3.618486
## 7   g 3.713864
## 8   h 3.809243
## 9   i 3.904621
## 10  j 4.000000
```

Ենթակառուցվածք սրանալու գործողությունների միջոցով հնարավոր է նաև նոր սյուններ ավելացնել փրված ուղղանկյուն-փոխարկային, ինչպես նաև հեռացնել՝

```
df[["New"]<-1:nrow(df)
df$one<-1
df
```

```
##      x          y New one
## 1 a 3.141593    1    1
## 2 b 3.236971    2    1
## 3 c 3.332350    3    1
## 4 d 3.427728    4    1
## 5 e 3.523107    5    1
## 6 f 3.618486    6    1
## 7 g 3.713864    7    1
## 8 h 3.809243    8    1
## 9 i 3.904621    9    1
## 10 j 4.000000   10    1
```

```
df$New<-NULL
df[["one"]<-NULL
df
```

```
##      x          y
## 1 a 3.141593
## 2 b 3.236971
## 3 c 3.332350
## 4 d 3.427728
## 5 e 3.523107
## 6 f 3.618486
## 7 g 3.713864
## 8 h 3.809243
## 9 i 3.904621
## 10 j 4.000000
```

*NULL*-ը հասրուկ արժեք է, որը ցույց է փալիս արժեքների գոյություն չունենալը:

Նեռացնենք մինչ *d*-ն հանդիպող փառերը պարունակող բոլոր փողերը, կամ, որ նույնն է, պահենք *d*-ից հետո եկող փառերը պարունակող բոլոր փողերը՝

```
df<-df[df$x>="d",]
df
```

```
##      x          y
## 4 d 3.427728
## 5 e 3.523107
## 6 f 3.618486
## 7 g 3.713864
## 8 h 3.809243
## 9 i 3.904621
## 10 j 4.000000
```

Ինչպես փեսնում եք ձախ կողմում պահվել է փողերի հին համարակալումը: Տողերի համարները գրնելու համար օգրագործենք *row.names()* ֆունկցիան՝

```
row.names(df)
```

```
## [1] "4" "5" "6" "7" "8" "9" "10"
```

Այս ֆունկցիայի միջոցով կարող ենք ինչպես ջնջել փողերի համարակալումը (իրականում այս դեպքում փեղի կունենա միայն վերահամարակալում սկսած 1-ից), կամ փողերին փալ ցանկացած անուններ՝

```
row.names(df)<-NULL
df
```

```
##      x      y
## 1 d 3.427728
## 2 e 3.523107
## 3 f 3.618486
## 4 g 3.713864
## 5 h 3.809243
## 6 i 3.904621
## 7 j 4.000000
```

```
row.names(df)<-LETTERS[11:(10+nrow(df))]
df
```

```
##      x      y
## K d 3.427728
## L e 3.523107
## M f 3.618486
## N g 3.713864
## O h 3.809243
## P i 3.904621
## Q j 4.000000
```

Նանենք նաև  $g$  փառին համապատասխան  $y$  արժեքը, ապա  $P$  փողում գրված փողն ամբողջությամբ, իսկ վերջում՝ երկրորդ սյունն ամբողջությամբ՝

```
df[df$x=="g", "y"]
```

```
## [1] 3.713864
```

```
df["P",]
```

```
##      x      y
## P i 3.904621
```

```
df[,2]
```

```
## [1] 3.427728 3.523107 3.618486 3.713864 3.809243 3.904621 4.000000
```

```
#or
```

```
df[,-1]
```

```
## [1] 3.427728 3.523107 3.618486 3.713864 3.809243 3.904621 4.000000
```

Վերջին երկու դեպքում **R**-ը փեսանելով որ վերադարձվում է ընդամենը մեկ սյուն, այն վերածեց համասեռ վեկտորի, այսինքն՝ կորցրեց չափողականություն հարկանիշը: Այս գործողությունից խուսափելու համար կարող ենք օգտագործել հետևյալ կանչը՝

```
df[,2,drop=FALSE]
```

```
##      y
## K 3.427728
## L 3.523107
## M 3.618486
## N 3.713864
## O 3.809243
## P 3.904621
```

```
## Q 4.000000
#or
df[,-1,drop=FALSE]

##           y
## K 3.427728
## L 3.523107
## M 3.618486
## N 3.713864
## O 3.809243
## P 3.904621
## Q 4.000000
```

Ինչպես տեսնում եք՝ քառակուսի փակագծերում փողերի և սյուների համար նախադրված փողերում (սփորակներից համապատասխանաբար ձախ և աջ) հնարավոր է գրել ինչպես փողերի և սյուների համարները՝ դրական և բացասական, այնպես և դրանց անունները, ինչպես նաև փրամաբանական գործողություններ:

### 2.2.16 Կառուցվածքի փոփոխություն

Ինչպես և տեսակների դեպքում էր հնարավոր կարարել փոխարկում՝ ինչպես բացահայտ այնպես և քողարկված, կառուցվածքները ևս հնարավոր է փոխարկել մեկը մյուսին: Վերը նշված բոլոր կառուցվածքների համար կան դրանց փոխարկող համապատասխան ֆունկցիաները՝ *as.vector()*, *as.factor()*, *as.matrix()*, *as.list()*, *as.data.frame()* և հարուկ ուղղանկյուն-փոխալ նապրիցի փոխարկող *data.matrix()* ֆունկցիան: Դիտարկենք սրանցից մի քանիսը:

Նորից դիտարկենք հետևյալ ցուցակը՝

```
l<-list(a=data.frame(x=letters[1:4], y=seq(2,3,length.out=4)),
        b=list(pi,3.5,4),c=matrix(1:6,ncol=3))
```

Ինչպես նշվեց՝ [ փակագծի միջոցով ենթակառուցվածք ստանալիս վերադարձնում է նույն տեսակի կառուցվածք ինչ սկզբնական կառուցվածքն է՝

```
l[1]
```

```
## $a
##   x           y
## 1 a 2.000000
## 2 b 2.333333
## 3 c 2.666667
## 4 d 3.000000
```

Ինչպես տեսնում ենք այս ցուցակը հեշտությամբ կարելի է ուղղանկյուն-փոխալի վերածել, հետևաբար կարող ենք օգտագործել *as.data.frame()* ֆունկցիան և կարարել նշված փոխարկումը՝

```
df1 <- as.data.frame(l[1])
df1
```

```
##   a.x      a.y
## 1   a 2.000000
## 2   b 2.333333
## 3   c 2.666667
## 4   d 3.000000
```

Ինչպես նշվել է՝ ցուցակների և ուղղանկյուն-փոխալների հիմնական տարբերությունն է, որ ցուցակները կարող են պահել տարբեր երկարությամբ վեկտորներ: Օրինակ, դիտարկենք երկու համասեռ վեկտոր որոնցից մեկը պահում է ուսանողների անունները, իսկ մյուսը պահում է գնահատականներ՝

```
students<-c("George Sr.", "Bill", "George", "Barack", "Donald")
grades<-c(12, 34, 54, 65)
length(students)
```

```
## [1] 5
```

```
length(grades)
```

```
## [1] 4
```

Ինչպես տեսնում ենք այս երկուսն ունեն փոքր երկարություններ և փոքր տեսակներ, հետևաբար դրանք կարող ենք պահել միայն ցուցակում՝

```
l<-list(students=students, grades=grades)
```

Նման ձևով սովորաբար պահվում են իրարից անկախ փոխադրյալները: Եթե գիտենք, որ փոխադրյալները համապատասխանում են փոխադրյալների, միայն վերջին ուսանողի գնահատականը դեռևս հայտնի չէ, ապա օգտագործելով *NA* արժեքը հնարավոր է այս երկու վեկտորները բերել նույն երկարության և ցուցակը վերածել ուղղանկյուն-փոխադրյալի՝

```
l[[2]][5]<-NA
l
```

```
## $students
## [1] "George Sr." "Bill"          "George"      "Barack"      "Donald"
##
## $grades
## [1] 12 34 54 65 NA
```

```
d<-as.data.frame(l, stringsAsFactors = FALSE)
```

```
d
```

```
##   students grades
## 1 George Sr.    12
## 2      Bill     34
## 3     George    54
## 4     Barack    65
## 5     Donald    NA
```

Այսպիսով պարզ է դառնում ուղղանկյուն-փոխադրյալի օգտագործման մեկ կարևոր օրինակ՝ երբ պետք է լինում պահել միևնույն սուբյեկտների (մարդկանց, մեքենաների, շենքերի և այլն) վերաբերյալ փոքր տեսակի փոխադրյալներ, ապա սուբյեկտները կարող են լինել ուղղանկյուն-փոխադրյալի փողեր, իսկ համապատասխան փոխադրյալի տեսակները՝ սյուներ: Եթե որևէ սուբյեկտի վերաբերյալ փոխադրյալից որևէ մեկը բացակայում է, ապա որպեսզի համապատասխան փոխադրյալը պարունակող սյունը չկարճանա, ավելացնում են *NA* արժեքը:

Նման կարարենք հակառակ գործողությունը՝ ուղղանկյուն-փոխադրյալը վերածենք ցուցակի և հեռացնենք *NA* պարունակող արժեքը՝

```
l1<-as.list(d)
l1$grades<-l1$grades[!is.na(l1$grades)]
l1
```

```
## $students
## [1] "George Sr." "Bill"          "George"      "Barack"      "Donald"
##
## $grades
## [1] 12 34 54 65
```

Եթե ուղղանկյունաձև կառուցվածքը բաղկացած է միայն նույն տեսակի արժեքներից, ապա այն կարելի է վերածել մատրիցի՝ *data.matrix()* ֆունկցիայի միջոցով՝



```
M<-data.frame(a=c("4","5","6","7"),b=7:4, stringsAsFactors = FALSE)
```

```
M
```

```
##   a b
## 1 4 7
## 2 5 6
## 3 6 5
## 4 7 4
```

```
m1<-data.matrix(M)
```

```
class(m1)
```

```
## [1] "matrix"
```

```
m1
```

```
##      a b
## [1,] 4 7
## [2,] 5 6
## [3,] 6 5
## [4,] 7 4
```

```
m2<-as.matrix(M)
```

```
m2
```

```
##      a  b
## [1,] "4" "7"
## [2,] "5" "6"
## [3,] "6" "5"
## [4,] "7" "4"
```

Ինչպես տեսանք նույնը հնարավոր է անել նաև *as.matrix()* ֆունկցիայի միջոցով, պարզապես վերջինս տեսակի բողարկված փոփոխություն կատարելիս սրացավ տեքստային մատրից, իսկ առաջինը՝ թվային:

### 2.2.17 Նարցեր

1. Ի՞նչ կվերադարձնի հետևյալ արտահայտությունը՝

```
TRUE & 3<2
```

#### Պատասխան

Այսպեղ կարևոր է հասկանալ գործողությունների կարարման հերթականությունը: & նշանն իրականում նախասահմանված ֆունկցիա է, որն ընդունում է որպես արգումենտներ դրա ձախ և աջ կողմերում գրված արտահայտությունները: Ուշադիր պետք է լինել, որ այն որպես իր երկրորդ արգումենտ ընդունում է իրենից աջ գրված ամբողջ արտահայտությունը, այսինքն՝ ոչ թե միայն 3-ը, այլ՝ 3 < 2 ամբողջությամբ՝

```
TRUE & 3<2
```

```
## [1] FALSE
```

```
#the same as
`&`(TRUE, 3<2)
```

```
## [1] FALSE
```

2. Ի՞նչ կվերադարձնի հետևյալ արտահայտությունը՝

```
0 & TRUE<2
```

#### Պատասխան

Այսպեղ նույնպես պետք է հասկանալ որպես ֆունկցիա, որն ընդունում է իր ձախ և աջ կողմերում գրված արժեքները որպես արգումենտներ: Ձախ կողմում գրված է թվային արժեք, իսկ & ֆունկցիան ընդունում է որպես արգումենտներ միայն փրամաբանական արժեքներ, հետևաբար փեղի է ունենում արժեքի տեսակի քողարկված փոփոխություն և, ինչպես գիտենք, 0-ն փոխվում է *FALSE*-ի: Աջ կողմում ևս փեղի է ունենում արժեքի փոփոխություն: Թվային համեմատություն կատարելու համար *TRUE* արժեքը փոխվում է 1-ի և համեմատվում 2-ի հետ:

```
`&`(0, TRUE<2)
```

```
## [1] FALSE
```

3. Ի՞նչ կվերադարձնի հետևյալ արտահայտությունը՝

```
x <- 5!=5
!x+1
```

#### Պատասխան

! նշանը ֆունկցիա է, որը որպես իր արգումենտ ընդունում է իրենից աջ գրված ամբողջ արտահայտությունը: Այսպեղ կարող է փոխադրություն ստեղծվել, որ ! կիրառված է միայն *x*-ի վրա, բայց իրականում այն կիրառված է ամբողջ աջ մասի վրա, հետևաբար նախ պետք է կատարել գումարման գործողությունը, ապա՝ կիրառել ժխտման գործողությունը՝

```
x <- 5!=5
x
```

```
## [1] FALSE
```

```
`!`(x+1) # !x+1
```

```
## [1] FALSE
```

```
# the same as
!(x+1)
```

```
## [1] FALSE
```

```
# NOT the same as
(!x)+1
```

```
## [1] 2
```

4. Ի՞նչ կվերադարձնի հետևյալ արտահայտությունը՝

```
x<-matrix(1:4, nrow=2, byrow = TRUE)
x>c(1,2)
```

#### Պատասխան

Քանի որ համեմատության մեջ գրնվող երկրորդ վեկտորն ավելի կարճ է քան մատրիցը (մատրիցի երկարությունը հավասար է իր տարրերի քանակին), ապա վեկտորը կրկնվում է այնքան անգամ մինչև դրա երկարությունը հավասարվի մատրիցի երկարությանը: Այսպեղ վեկտորը կրկնվելով ևս մեկ անգամ կստանանք ճշգրիտ մատրիցի երկարությունը, դրա համար ծրագրի կողմից զգուշացում չի տրվի: Երբ վեկտորի երկարությունը հավասարվում է մատրիցի երկարությանը, ապա վեկտորը վերածվում է մատրիցի՝ տարրերի տեղաբաշխումը կատարելով այուներով (*byrow=FALSE*), քանի որ այդպիսինն է *matrix()* ֆունկցիայի լռելյալ վարքագիծը, և օգտագործելով առաջին մատրիցի տողերի քանակը: Այս ամբողջից հետո կկատարվի համեմատությունը:

```
x<-matrix(1:4, nrow=2, byrow = TRUE)
x
```

```
##      [,1] [,2]
## [1,]    1    2
```

```
## [2,] 3 4
matrix(rep(1:2, times=2), nrow=2, byrow = FALSE)

##      [,1] [,2]
## [1,] 1 1
## [2,] 2 2
x>matrix(rep(1:2, times=2), nrow=2, byrow = FALSE)

##      [,1] [,2]
## [1,] FALSE TRUE
## [2,] TRUE TRUE
# the same as
x>c(1,2)

##      [,1] [,2]
## [1,] FALSE TRUE
## [2,] TRUE TRUE
```

### 2.2.18 Խնդիրներ

1. Տրված է հերկյալ համաստեղ վեկտորը՝

```
a<-c(3,4,3,2,1,-1,6,-1,-2)
```

Լուծել հավասարումների հերկյալ համակարգը՝

$$\begin{cases} 3x_1 + 4x_2 + 3x_3 = 6 \\ 2x_1 + x_2 - x_3 = 10 \\ 6x_1 - x_2 - 2x_3 = -4 \end{cases}$$

Լուծումները ներկայացնել հարյուրերորդական ճշտությամբ:

**Լուծում**

```
a<-c(3,4,3,2,1,-1,6,-1,-2)
b<-c(6,10,4)
A<-matrix(a,byrow = TRUE,nrow=3)
x<-solve(A)%*%b #or solve(A,b)
x<-round(x,2)
x
```

```
##      [,1]
## [1,] -0.10
## [2,] 5.27
## [3,] -4.93
```

2. Ստեղծել հերկյալ համաստեղ վեկտորները՝

```
Continents<-c("Europe", "Australia", "Asia",
              "America", "Africa", "Antarctica")

x<-c(USA=120, Europe=320, Australia=360, Japan=450,
     Asia=680,America=200, France=80,Armenia=600,Africa=250)
x
```

```
##      USA      Europe Australia      Japan      Asia      America      France
##      120      320      360      450      680      200      80
##  Armenia      Africa
##      600      250
```

Մայրցամաքների համար, որոնց արժեքը չի գերազանցում 350-ը, մեծացրեք արժեքները 50%-ով: Փոփոխությունները կապարեք փոփոխված համասեռ  $x$  վեկտորի մեջ: Նանք փոփոխված արժեքները իրենց մայրցամաքների անունների հետ և պահեք այդ փոփոխված  $z$  վեկտորում:

### Լուծում

```
y<-names(x)%in%Continents & x<=350
x[y]<-x[y]*1.5
```

```
z<-x[y]
z
```

```
## Europe America Africa
##      480      300      375
```

### 3. Ստեղծեք հեղուկալ համասեռ վեկտորը`

```
set.seed(1)
n<-100
X<-c(sample(c(NaN,NA,1),replace=TRUE, size=n),rep("a",n))
X[1:10]
```

```
## [1] "NaN" NA      NA      "1"      "NaN" "1"      "1"      NA      NA      "NaN"
```

Նաշվեք  $NA$ -ների քանակը (ճշգրիտ  $NA$ -ների), ինչպես նաև  $NaN$ -երի քանակը:

### Լուծում

Քանի որ համասեռ վեկտորում ներառված են նաև փոփոխ, ապա այդ վեկտորի տեսակը տեքստային է, այսինքն՝ թվային արժեքների տեսակի քողարկված փոփոխություն է փոփոխվել և դրանք վերածվել են տեքստի, հետևաբար  $NaN$ -երը նույնպես տեքստի են վերածվել: Այն թվերի վերածելու համար պետք է հեռացնենք փոփոխությունը: Տրված վեկտորից ստանանք ենթավեկտոր, որը կպարունակի միայն “ $NaN$ ” տեքստային արժեքները (իրականում կմնան նաև  $NA$  արժեքները, քանի որ  $R$ -ը չգիտի թե դրանք հավասար են “ $NaN$ ” տեքստային արժեքին, թե՛ ոչ):

```
sum(is.na(X)) #number of exact NAs
```

```
## [1] 38
```

```
X<-as.numeric(X[X=="NaN"])
X
```

```
## [1] NaN NA NA NaN NA NA NaN NaN NaN NA NA NA NaN NA NaN NaN NA
## [18] NaN NA NA NA NA NA NaN NaN NA NA NA NA NaN NA NA NA NaN
## [35] NaN NaN NaN NA NA NA NaN NA NaN NA NaN NA NaN NA NA NA NA
## [52] NA NA NA NaN NaN NaN NaN NaN NaN NaN NA NA NA NA
```

```
sum(is.nan(X))
```

```
## [1] 27
```

### 4. Դիտարկենք *airquality* ուղղանկյունաձև փոփոխությունները, որում պահվում են Նյու Յորքում մի քանի ամիսների ընթացքում կապարված մթնոլորտային չափումների արդյունքները: Մտորել գրված հարցերին պատասխանելու համար գրել մեկ փողանոց հրամաններ:

```
dat<-airquality
dim(dat)
```

```
## [1] 153 6
```

```
head(dat)
```

```
##   Ozone Solar.R Wind Temp Month Day
## 1    41    190  7.4  67     5    1
## 2    36    118  8.0  72     5    2
## 3    12    149 12.6  74     5    3
## 4    18    313 11.5  62     5    4
## 5    NA     NA 14.3  56     5    5
## 6    28     NA 14.9  66     5    6
```

- Ստեղծել նոր սյուն *Temp.C*, որի մեջ կպահվի ջերմաստիճանը Ֆելսիուսի սանդղակով: *Temp* սյան մեջ ջերմաստիճանը պահված է Ֆարենհայթի սանդղակով: Երկու սանդղակների միջև կապը հետևյալն է՝

$$C = \frac{5}{9}(F - 32).$$

Տվյալները պահելու համար օգտագործել փաստորդական ճշգրտությունը:

Ներագա բոլոր հաշվարկներում օգտագործել Ֆելսիուսի ջերմաստիճանը:

**Լուծում**

```
dat$Temp.C <- round((dat$Temp-32)*5/9, digits=1)
head(dat)
```

```
##   Ozone Solar.R Wind Temp Month Day Temp.C
## 1    41    190  7.4  67     5    1    19.4
## 2    36    118  8.0  72     5    2    22.2
## 3    12    149 12.6  74     5    3    23.3
## 4    18    313 11.5  62     5    4    16.7
## 5    NA     NA 14.3  56     5    5    13.3
## 6    28     NA 14.9  66     5    6    18.9
```

- Նաշվել մայիս ամսվա միջին ջերմաստիճանը:

**Լուծում**

```
mean(dat[dat$Month==5, "Temp.C"])
```

```
## [1] 18.63226
```

- Գտնել փարվա ամենաբարձր ջերմաստիճանին համապատասխանող օրը, ամիսը և ջերմաստիճանը (նշված հերթականությամբ):

**Լուծում**

```
dat[dat$Temp.C==max(dat$Temp.C), c("Day", "Month", "Temp.C")]
```

```
##   Day Month Temp.C
## 120  28     8   36.1
```

- Դիտարկելով միայն օրերը, երբ Ֆելսիուսի ջերմաստիճանը եղել է 30-ից բարձր, հաշվել քանու միջին արագությունը և փայլ բոլոր այն օրերը (ամսվա, ջերմաստիճանի և քանու արագության հետ մեկտեղ), երբ քանու արագությունը գերազանցում է այդ օրերի համար հաշված քանու միջին արագությանը:

**Լուծում**

```
dat[dat$Temp.C>30 & dat$Wind>mean(dat[dat$Temp.C>30, "Wind"]),
     c("Day", "Month", "Temp.C", "Wind")]
```

```
##      Day Month Temp.C Wind
## 40    9     6   32.2 13.8
## 41   10    6   30.6 11.5
## 42   11    6   33.9 10.9
## 43   12    6   33.3  9.2
## 71   10    7   31.7  7.4
## 75   14    7   32.8 14.9
## 89   28    7   31.1  7.4
## 100   8     8   32.2 10.3
## 101   9     8   32.2  8.0
## 102  10    8   33.3  8.6
## 120  28    8   36.1  9.7
## 128   5     9   30.6  7.4
```

### 2.2.19 Ամփոփում

- Type Coercion
- Missing Values
- Subsetting
- Attributes
- Vector, Factor, Matrix, List, Data Frame
- Partial Matching

#### Functions

- *class()*
- *length()*
- *nchar()*
- *c()*
- *vector()*
- *rep()*
- *seq()*, *seq\_len()*, *seq\_along()*
- *as.logical()*, *as.integer()*, *as.numeric()*, *as.complex()*, *as.character()*
- *sum()*, *round()*, *min()*, *max()*, *mean()*, *sqrt()*
- *which()*
- *names()*
- *Sys.time()*
- *attributes()*, *attr()*
- *matrix()*
- *dim()*, *nrow()*, *ncol()*
- *rbind()*, *cbind()*
- *det()*, *solve()*, *t()*
- *list()*

- `factor()`, `table()`
- `data.frame()`, `head()`, `tail()`
- `row.names()`
- `as.vector()`, `as.factor()`, `as.matrix()`, `as.data.frame()`, `data.matrix()`, `as.list()`

## 2.3 Կարգավորման կառուցվածքներ (Control Structures)

### 2.3.1 Տրամաբանություն (Logic)

```
TRUE & c(TRUE,FALSE,FALSE)
```

```
## [1] TRUE FALSE FALSE
```

```
TRUE && c(TRUE,FALSE,FALSE)
```

```
## [1] TRUE
```

```
TRUE | c(TRUE,FALSE,FALSE)
```

```
## [1] TRUE TRUE TRUE
```

```
TRUE || c(TRUE,FALSE,FALSE)
```

```
## [1] TRUE
```

```
#All AND operators are evaluated before OR operators.
```

```
5 > 8 || 6 != 8 && 4 > 3.9
```

```
## [1] TRUE
```

```
#identical(), isTRUE()
```

```
#exclusive OR. If one argument evaluates to TRUE and one argument evaluates to #FALSE, then this function will return TRUE, otherwise it will return FALSE.
```

```
xor(5 == 6, !FALSE)
```

```
## [1] TRUE
```

```
#
```

```
which(1:10>7)
```

```
## [1] 8 9 10
```

```
#
```

```
ints<-sample(10)
```

```
any(ints<0)
```

```
## [1] FALSE
```

```
all(ints>0)
```

```
## [1] TRUE
```

**2.3.2 Գործողությունների իրականացման հերթականության կարգավորում (Control Flow)**

Այս կառուցվածքի միջոցով կատարվում է արտահայտությունների պայմանական կատարում՝ կախվախ փրված պայմանի ճշմարտացիությունից: Եթե փրված պայմանը ճիշտ է կատարվում է որևէ գործողություն, իսկ եթե սխալ է, այդ գործողությունը չի կատարվում կամ, անհրաժեշտության դեպքում, կատարվում է մեկ այլ գործողություն: Ննարավոր է նաև փրված պայմանի սխալ լինելու դեպքում ևս մեկ պայման ստուգել ու դրա ճշմարտացիությունից ելնելով կատարել գործողություններ: Դիպարկենք երկու օրինակ: (Ուշադրություն դարձրեք, որ else բառը գրված է դրան նախորդող ձևավոր փակագծի հետ մեկ փողում: Եթե այն գրված լիներ հաջորդ փողում, փրկի կունենար սխալ:)

```
x<-4
if(x>3){
  print("The value is greater than 3")
} else {
  print("The value is less than or equal to 3")
}
```

```
## [1] "The value is greater than 3"
```

```
x<-2.5
if(x>3){
  print("The value is greater than 3")
} else if(x>2){
  print("The value is in (2,3]")
} else{
  print("The value is less than or equal to 2")
}
```

```
## [1] "The value is in (2,3]"
```

Ննարավոր է նաև այս կառուցվածքի միջոցով կատարել վերագրում՝

```
x<-12
y<-if(x%%2==0){
  "A Prime"
} else{
  "Not a Prime"
}
y
```

```
## [1] "A Prime"
```

If արտահայտությունում պայման գրելիս պետք է ուշադիր լինել, որ վերադարձվող փրամաբանական արժեքը պարունակվի 1 (և ոչ ավել) երկարությամբ վեկտորում:

```
x<-1:2
x%%2==0
```

```
## [1] FALSE TRUE
```

```
y<-if(x%%2==0){
  x
}
```

```
## Warning in if (x%%2 == 0) {: the condition has length > 1 and only the
## first element will be used
```

```
y
```

```
## NULL
```



Բերված օրինակում մենք որպես պայման գրել ենք վեկտոր, որը ոչ թե մեկ այլ երկու պրամաբանական արժեք է վերադարձնում, հետևաբար **R**-ը վերցնում է այդ վեկտորի միայն առաջին արժեքը: Քանի որ պայմանի վերադարձրած արժեքի երկարությունը պետք է 1 լինի, ապա վեկտորների հետ աշխատելիս օգտագործվում են կրկնակի նշանները, որոնք հենց վերցնում են միայն վեկտորի առաջին կոորդինատը: Ստորև գրված երկու ծրագրային իրագործումները կախարում են նույն գործողությունը, բացառությամբ, որ երկրորդն այլևս զգուշացում չի փախի՝

```
x<-c(3,4)
y<-c(1,5)
if(x>y & TRUE) print(x)
```

```
## Warning in if (x > y & TRUE) print(x): the condition has length > 1 and
## only the first element will be used
```

```
## [1] 3 4
```

```
x<-c(3,4)
y<-c(1,5)
if(x>y && TRUE) print(x)
```

```
## [1] 3 4
```

Գոյություն ունի նաև նախասահմանված ֆունկցիա, որը կախարում է այս կառուցվածքի գործառույթները և ավելի հարմար է օգտագործել պարզ պայմանների առկայության դեպքում՝

```
x<-12
y<-ifelse(x%%2==0,"A Prime", "Not a Prime")
y
```

```
## [1] "A Prime"
```

Այս ֆունկցիան երբեմն կարող է ավելի օգտակար լինել, քան համապատասխան կառուցվածքը, քանի որ դրանում որպես պայման կարելի է փոխանցել մեկից ավելի երկարություն ունեցող վեկտոր՝

```
x<-1:2
y<-ifelse(x%%2==0,"A Prime", "Not a Prime")
y
```

```
## [1] "Not a Prime" "A Prime"
```

### 2.3.3 Շրջապարույրի կառուցվածքներ (Loop Structures)

#### 2.3.3.1 For Loop

Շրջապարույրի համար նախատեսված for կառուցվածքը վերցնում է կրկնման համար նախատեսված փոփոխականը և դրան հաջորդաբար վերագրում է արժեքներ որևէ հաջորդականությունից (օրինակ՝ վեկտորից):

```
for(i in 1:5){
  print(i)
}
```

```
## [1] 1
```

```
## [1] 2
```

```
## [1] 3
```

```
## [1] 4
```

```
## [1] 5
```

Այսպեղ կրկնման համար նախապես սկսած փոփոխականը  $i$ -ն է, որը հաջորդաբար ընդունում է 1:5 վեկտորում առկա արժեքները, իսկ կառուցվածքի ներսում, ամեն քայլին կատարվում են որոշակի գործողություններ: Այս կառուցվածքը կարող է ունենալ փարբեր գրելաձևեր՝ նույն գործողությունը կատարելու համար՝

```
x<-c(12,34,55,64)
for(i in 1:length(x)){
  print(x[i])
}
```

```
## [1] 12
## [1] 34
## [1] 55
## [1] 64
```

```
x<-c(12,34,55,64)
for(i in seq_len(length(x))){
  print(x[i])
}
```

```
## [1] 12
## [1] 34
## [1] 55
## [1] 64
```

```
x<-c(12,34,55,64)
for(i in seq_along(x)){
  print(x[i])
}
```

```
## [1] 12
## [1] 34
## [1] 55
## [1] 64
```

```
x<-c(12,34,55,64)
for(i in x){
  print(i)
}
```

```
## [1] 12
## [1] 34
## [1] 55
## [1] 64
```

Նսարավոր է նաև ներդնել for շրջապտույտի կառուցվածքները՝

```
x<-matrix(1:4,ncol=2)
x
```

```
##      [,1] [,2]
## [1,]   1   3
## [2,]   2   4
```

```
for(i in seq_len(nrow(x))){
  for(j in seq_len(ncol(x))){
    print(x[i,j])
  }
}
```

```
## [1] 1
```

```
## [1] 3
## [1] 2
## [1] 4
```

Նաճախ սա խնդիրներ լուծելու ոչ ամենաարդյունավետ մեթոդն է, սակայն երբեմն այն միակ կիրառելին է: Եթե հնարավորություն կա՝ պետք է խուսափել շրջապտույտի ներդրված կառուցվածքներից: Ներառյալ խնդիրներ են շրջապտույտի կառուցվածքով գրված լուծումները վերածել առանց շրջապտույտի կառուցվածքով լուծումների: Դիտարկենք հետևյալ օրինակը, երբ պետք է մաքրիցի բոլոր փարրերը փոխարինել 0-ով՝

```
x<-matrix(1:4,ncol=2)
x
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
for(i in seq_len(nrow(x))){
  for(j in seq_len(ncol(x))){
    x[i,j]<-0
  }
}
x
```

```
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0
```

Սա հնարավոր է լուծել մեկ փողով՝ առանց որևէ շրջապտույտի,

```
x<-matrix(1:4,ncol=2)
x
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
```

```
x[,]<-0
x
```

```
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0
```

### 2.3.3.2 While Loop

Շրջապտույտի մյուս կառուցվածքն է while կառուցվածքը:

```
i<-0
while(i<5){
  print(i)
  i<-i+1
}
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] 3
## [1] 4
```

```
print(i)
```

```
## [1] 5
```

Նախորդի համեմատությամբ կան մի քանի փոփոխություններ: Նախ կրկնման փոփոխականը պետք է սկզբնաբժեք սրանա կառուցվածքից դուրս, իսկ կառուցվածքի ներսում դրա արժեքը պետք է փոխվի, քանի որ հակառակ դեպքում փեղի կունենա անվերջ շրջապտույտ: Այսպիսով՝ այս կառուցվածքն ավելի վրանգավոր է քանի որ ծրագրավորողն ինքը պետք է կարգավորի շրջապտույտի սկիզբը և քայլը, ապահովելու համար շրջապտույտի ավարտը: Մյուս կողմից էլ դրանք առավելություն ունեն քանի որ հնարավորություն են տալիս միաժամանակ մի քանի պայմաններ ստուգելու: Դիփարկենք պարահական թափառման հետևյալ օրինակը, երբ շարժվողը դուրս է գալիս որևէ կետից և 0.5 հավանականությամբ մեկ քայլ կատարում աջ կամ ձախ (ուղղությունը կարող է որոշվել, օրինակ, արդար մետաղադրամ նետելիս): Թափառումն ավարտվում է եթե շարժվողը հասնում է երկու՝ ամեն կողմում մեկական, նպատակներից որևէ մեկը:

```
z<-7
while(z>=0 & z<=10){
  if(rbinom(n=1,size=1,prob=0.5)==1) z<-z+1
  else z<-z-1
}
print(z)
```

```
## [1] 11
```

Նիշենք, որ փրամաբանական գործողությունները կատարվում են ձախից աջ:

### 2.3.4 Repeat, Next, Break, Return

*Repeat* հրամանը ստեղծում է անվերջ շրջապտույտ, որը կարող է կասեցվել միայն *break*՝ դադար, հրամանի միջոցով:

```
i<-0
repeat{
  i<-i+1
  if(i==5) break
}
print(i)
```

```
## [1] 5
```

*Next* հրամանն օգտագործվում է շրջապտույտի ժամանակ կրկնվող փոփոխականի որևէ արժեք բաց թողնելու համար: Ենթադրենք ուզում ենք տպել մինչ 10-ն առկա բոլոր գույգ թվերը՝ բացառությամբ 6-ի և 8-ի՝

```
for(i in 1:10){
  if(i in c(6,8)) next
  if(i%2==0) print(i)
}
```

```
## [1] 2
```

```
## [1] 4
```

```
## [1] 10
```

Շրջապտույտը դադարեցնելու համար կարող ենք օգտագործել ինչպես *break* այնպես և *return* հրամանները: Այս վերջինը հաճախ օգտագործվում է ֆունկցիաների սահմանման մեջ, բայց կարող է օգտակար լինել նաև շրջապտույտը դադարեցնելու համար: Ստորև գրված ծրագիրը գրնում է 20-ից հետո 16-ին բաժանվող առաջին թիվը՝

```
i<-20
while(TRUE){
```

```

if(i%%16==0) return(i)
i<-i+1
}
i

```

```
## [1] 32
```

### 2.3.5 Խնդիրներ

1. Դիտարկենք *airquality* ուղղանկյունաձև փյուռները, որում պահվում են Նյու Յորքում մի քանի ամիսների ընթացքում կարարված մթնոլորտային չափումների արդյունքները: Մեր նպատակն է ուսումնասիրել ջերմաստիճանի (*Temp*) բաշխումը:

Ջերմաստիճանի միջակայքը՝ նվազագույնից առավելագույն հասնելու հարվածը, բաժանում ենք երեք հավասար մասերի և հաշվում յուրաքանչյուր հարվածում ընկնող ջերմաստիճանի չափումների քանակները: Առանձնացնում ենք այն հարվածը, որն ամենաքիչ քանակով կետերն է պարունակում և *airquality* կառուցվածքից հեռացնում ենք բոլոր այն փողերը որոնց ջերմաստիճանն ընկնում է այդ հարվածում:

#### Լուծում

Նախ փանք կարճ լուծում, որը, սակայն, դժվար կլինի ընդհանրացնել 3-ից մեծ թվով հարվածների համար:

```

rm(list=ls())
dat<-airquality
x<-"Temp"

m<-min(dat[,x])
M<-max(dat[,x])
d<-(M-m)/3
M1<-m+d
M2<-m+2*d
n1<-sum(m<=dat[,x] & dat[,x]<=M1)
n2<-sum(M1<dat[,x] & dat[,x]<=M2)
n3<-sum(M2<dat[,x] & dat[,x]<=M)

if(min(c(n1,n2,n3))==n1){
  xmin<-m
  xmax<-M1
} else if(min(c(n1,n2,n3))==n2){
  xmin<-M1
  xmax<-M2
} else {
  xmin<-M2
  xmax<-M
}

dat0<-dat[!(xmin<=dat[,x] & dat[,x]<=xmax),]
dim(dat)

## [1] 153 6

dim(dat0)

## [1] 121 6

```

Նաջորդ լուծումը փրկված  $N$ -ի համար ջերմաստիճանի միջակայքը բաժանում է  $N$  հավասար մասերի և հաշվում յուրաքանչյուր հարվածում ընկած ջերմաստիճանի չափումների քանակները: Նվազագույն քանակով

ջերմաստիճանի չափումներ պարունակող հարվածին համապատասխան օրերը հեռացվում են *airquality* ուղղանկյուն քվյալներից:

```
rm(list=ls())

dat<-airquality
X<-"Temp"

N<-8

Y <- dat[,X]

Int <- seq(min(Y),max(Y),length.out = N+1)

#x<-Y[1]

Indices <- vector()
for(x in Y){
  if(all(Int<=x)) i<-N+1
  else i<- which(Int>x)[1]
  Indices <- c(Indices,i)
}
table(Indices)

## Indices
##  2  3  4  5  6  7  8  9
## 11 10 15 25 35 30 15 12

i0<-as.numeric(names(table(Indices)[table(Indices)==min(table(Indices))]))
dat0<-dat[!(Y>=Int[i0-1] & Y<=Int[i0]),]
```

### 2.3.6 Անփոփում

- Loop Structures

Functions

- *any()*, *all()*
- *identical()*, *isTRUE()*
- *xor()*
- *ifelse()*

## 2.4 Ֆունկցիաներ

### 2.4.1 Ֆունկցիայի սահմանումը

Այն ամենն ինչ գոյություն ունի **R**-ում օբյեկտ է, այն ամենն ինչ կարարվում է՝ ֆունկցիայի կանչ (Ջոն Չեմբերս):

**R**-ում ֆունկցիաները կարարում են նույն դերը ինչ մաթեմատիկայում՝ դրանք համապատասխանություն են ստեղծում օբյեկտների միջև, այն է՝ դուք դրանց փոխանցում եք որևէ օբյեկտ, որը կոչվում է այդ ֆունկցիայի արգումենտ, իսկ այն ձեզ վերադարձնում է մեկ այլ օբյեկտ, որը կոչվում է արժեք: Ննարավոր են նաև դեպքեր, երբ ֆունկցիան ոչ մի արժեք չի վերադարձնում, պարզապես կանչելիս այն կարարում է որոշակի գործողություն: Ննարավոր է նաև, որ ֆունկցիան արգումենտներ չունենա, այսինքն՝ գործողություն կարարելու

համար կամ արժեք վերադարձնելիս այն օգտագործողի կողմից օբյեկտների ներմուծման կարիք չունի: Երբ գործողությունների որոշակի շարք անընդհար օգտագործվում է, ապա նպատակահարմար է դա սահմանել ֆունկցիայի տեսքով և վերաօգտագործման ժամանակ ընդամենը կանչել այդ ֆունկցիան: Դիտարկենք ֆունկցիայի առաջին օրինակը, որում տրված թվի համար հաշվում է դրա քառակուսի արմատը: Մա նման է լինելու նախասահմանված `sqrt()` ֆունկցիային, միակ տարբերությամբ, որ վերջինս իրական թիվ ստանալու դեպքում իրական թիվ էր վերադարձնում, իսկ մեր ֆունկցիան միշտ կոմպլեքս թիվ է վերադարձնելու:

```
Sqrt <- function(comp){
  comp<-as.complex(comp)
  sqrt(comp)
}
```

```
Sqrt(-1)
```

```
## [1] 0+1i
```

Ինչպես տեսնում եք՝ արժեք վերադարձնելու `return` հրամանն առկա չէ ֆունկցիայի սահմանման մեջ: Այս դեպքում ամենավերջում գրված արժեքն ինքնաբերաբար վերադարձվում է: `Return` հրամանի կիրառումն առավելապես օգտակար է լինում երբ վերադարձվող արժեքը պայմանից է կախված: Օրինակ, ստորև գրված ֆունկցիան նույնպես հաշվում է տրված թվի արմատը, իսկ բացասական թիվ ստանալու դեպքում այն պարզապես վերադարձնում է հաղորդագրություն, որ նշված թվի արմատը հնարավոր չէ հաշվել՝ ի տարբերություն նախասահմանված `sqrt()` ֆունկցիայի, որը նման դեպքերում դադարեցնում էր հրամանի ի կատար ածումը:

```
Sqrt_ <- function(x) {
  if(x<0) return("A Negative number is provided!")
  return(sqrt(x))
}
```

```
Sqrt_(-5)
```

```
## [1] "A Negative number is provided!"
```

```
Sqrt_(5)
```

```
## [1] 2.236068
```

Ինչպես տեսնում եք՝ `if` հրամանից հետո `else` հրամանը գրելու անհրաժեշտություն չկա, քանի որ առաջին հանդիպած `return` հրամանի իրագործումից հետո ֆունկցիայի գործողությունը դադարեցվում է:

Ցանկացած, այդ թվում և նախասահմանված, ֆունկցիայի կառուցվածքը ստանալու համար պարզապես պետք է տպել դրա անունը՝ առանց արգումենտների համար նախադրեալ փակագծերի՝

```
Sqrt
```

```
## function(comp){
##   comp<-as.complex(comp)
##   sqrt(comp)
## }
```

Ֆունկցիանները կարելի է այնպես սահմանել, որ դրանց որոշ արգումենտներն ունենան լռելյալ արժեքներ, այսինքն՝ հնարավոր լինի ֆունկցիան կանչել առանց այդ արգումենտներին արժեքներ տալու: Ներկայի ֆունկցիան վերցնում է երկու արգումենտ և վերադարձնում է առաջին արգումենտը՝ երկրորդ արգումենտով աստիճան բարձրացրած՝

```
Power <- function(base, exp=2){
  return(base^exp)
}
```

Երկրորդ արգումենտն ունի լռելյալ արժեք, որը երկուս է: Այսինքն՝ եթե երկրորդ արգումենտին արժեք չփոխանցենք, ապա կվերադարձվի առաջին արգումենտի քառակուսին: Ֆունկցիան կանչելիս և դրան

արգումենտներ փոխանցելիս կարելի է արգումենտի անունը, որին փոխանցվում է արժեքը, չնշել: Այդ դեպքում Կրեյի կունենա արգումենտների հավասարեցում փոխանցված արժեքներին՝ դիրքի միջոցով: Օրինակ՝

```
Power(10)
```

```
## [1] 100
```

Այս դեպքում փոխանցված արժեքը փոխանցվում է առաջին արգումենտին, իսկ երկրորդ արգումենտը կարող է և արժեք չստանալ, քանի որ այն լռելյալ արժեք ունի: Արգումենտների արժեքները նշելուց հետո հնարավոր է արգումենտների դիրքերը նշել իրենց սահմանված դիրքերին չհամապատասխանող՝

```
Power(base=5,exp=3)
```

```
## [1] 125
```

```
Power(exp=3,base=5)
```

```
## [1] 125
```

Ննարավոր է նաև անունների մասնակի նշում կամ միայն մի արգումենտի նշում անունով, իսկ մյուսների նշում դիրքով՝

```
Power(base=16,e=1.5)
```

```
## [1] 64
```

```
#or, using the names and the positions
```

```
Power(base=16,1.5)
```

```
## [1] 64
```

Արգումենտի նշման երեք ձևերը՝ անունով, դիրքով և մասնակի անունով միասին կիրառելիս Կրեյի ունի կարարման հերթականությունը՝ 1. անունով, 2. մասնակի անունով և ամենավերջում այն արգումենտները որոնք դեռևս կանչված չեն կարող են դիմվել դիրքով:

Այժմ ստեղծենք ֆունկցիա, որը հաշվում է փոխանցված մատրիցի փողերի կամ սյուների միջին արժեքներից բաղկացած վեկտոր:

```
Average <- function(x, bycol = TRUE){
  if(!bycol) x<-t(x)
  nc <- ncol(x)
  av_values<-vector(mode="numeric", length=nc)
  for(i in seq_len(nc)){
    av_values[i]<-mean(x[,i])
  }
  av_values
}
```

```
M1 <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12), ncol=3)
```

```
Average(M1)
```

```
## [1] 2.5 6.5 10.5
```

```
Average(M1, bycol = FALSE)
```

```
## [1] 5 6 7 8
```

```
Average(t(M1), bycol = FALSE)
```

```
## [1] 2.5 6.5 10.5
```



Ֆունկցիայի արգումենտների անունները սրանալու համար կարելի է օգտագործել նախասահմանված `args()` ֆունկցիան՝

```
args(Average)
```

```
## function (x, bycol = TRUE)
## NULL
```

Սրացվեց, որ ֆունկցիային որպես արգումենտ հնարավոր է ֆունկցիա փոխանցել: Ավելի խորությամբ ուսումնասիրենք այս հատկությունը: Սահմանենք ֆունկցիա, որը սրանում է երեք արգումենտ՝ վեկտոր, որը պարունակում է փվյալները, երկարությունը, որով պետք է կտրել նախորդ վեկտորը և ֆունկցիա, որը պետք է կիրառել կտրված վեկտորի վրա:

```
Statistic <- function(func,x,n){
  func(x[1:n])
}
```

```
x<-1:100
```

```
Statistic(mean,x=x,n=10)
```

```
## [1] 5.5
```

```
Statistic(sd,x,25)
```

```
## [1] 7.359801
```

```
Statistic(func=max,x=x,n=98)
```

```
## [1] 98
```

**R**-ում հնարավոր է ֆունկցիան փոխանցել որպես արգումենտ անգամ եթե այն դեռևս սահմանված չէ և սահմանվում է հենց օգտագործելիս: Նախապես չսահմանված ֆունկցիան կոչվում է անանուն ֆունկցիա (anonymous function): Դիտարկենք հետևյալ օրինակը՝

```
Statistic(function(x){x^2},x=x,n=5)
```

```
## [1] 1 4 9 16 25
```

Մեր սահմանած `Statistic()` ֆունկցիան կիրառում է փրված ֆունկցիան փրված վեկտորի վրա՝ վերջինս կտրելուց հետո: Վերը նշված օրինակում մենք որպես արգումենտ փոխանցում ենք քառակուսի բարձրացնող ֆունկցիան, որը, սակայն, դեռևս սահմանված չէր և սահմանվում է `Statistic()` ֆունկցիայի արգումենտում և միայն այդպիսով գոյություն ունի:

#### 2.4.2 Նրամանի կատարում ըստ անհրաժեշտության (Lazy Evaluation)

Ֆունկցիայի արգումենտների վերագրումը կատարվում է ըստ անհրաժեշտության: Դիտարկենք հետևյալ օրինակը՝

```
f <- function(a,b){
  a+2
}
f(10)
```

```
## [1] 12
```

Թեև վերը նշված ֆունկցիայում թեև մենք արժեք չենք վերագրում `b` արգումենտին ու այն չունի լռելյալ արժեք, բայց, այնուամենայնիվ, սխալ պեղի չի ունենում և ֆունկցիան արժեք է վերադարձնում: Պատճառն այն է, որ ֆունկցիան գործողություն կատարելու համար չունի `b`-ի կարիքը, այդ պատճառով էլ այն չի կանչում `b`-ի արժեքը: Սա կոչվում է հրամանի կատարում ըստ անհրաժեշտության:

### 2.4.3 Բազմակերպարգումենտը (Ellipsis or dot-dot-dot)

Դիտարկենք նախասահմանված `paste()` ֆունկցիան, որը փրկված վեկտորները միավորում է մեկ փեքսպային փոփոխականի մեջ: Օրինակ՝

```
paste("Hello", "World", "of", "R")
```

```
## [1] "Hello World of R"
```

Դիտարկենք այս ֆունկցիայի արգումենտները՝

```
args(paste)
```

```
## function (... , sep = " ", collapse = NULL)
```

```
## NULL
```

Ինչպես տեսնում ենք՝ առաջին արգումենտի տեղում գրված է բազմակերպը՝ `...`: Դա նշանակում է, որ կարող ենք փոխանցել ցանկացած քանակով վեկտորներ և դրանք կմիավորվեն մեկ փեքսպային փոփոխականի մեջ: Բազմակերպով նշված արգումենտը սովորաբար ամենավերջին արգումենտն է լինում, ինչը տեղի չունի վերը նշված `paste()` ֆունկցիայի դեպքում: Եթե բազմակերպ արգումենտից հետո այլ արգումենտներ են սահմանված, ապա դրանք անպայմանորեն պետք է ունենան լռելյալ արժեքներ և դրանց դիմելիս անունները պետք է նշվեն ամբողջությամբ: Ներկաբար բոլոր այն ֆունկցիաներում որտեղ առաջին արգումենտը բազմակերպ է բոլոր արգումենտների անունները պետք է նշվեն ամբողջությամբ՝

```
paste("a", "b", sep=":")
```

```
## [1] "a:b"
```

```
paste("a", "b", se=":")
```

```
## [1] "a b :"
```

Երկրորդ հրամանը չփրկեց նույն արդյունքը ինչ առաջինը քանի որ մենք արգումենտի անունը մասնակի ենք նշել այդ պարճառով դրա անունն անպետք է, իսկ դրա արժեքը հասկացվել է որպես հերթական վեկտոր, որը պետք է միացնել նախորդներին:

Այժմ տեսնենք թե ինչպես կարող ենք դիմել բազմակերպում նշված արգումենտներին (`unpack dot-dot-dot arguments`), երբ պետք է լինում նման արգումենտով ֆունկցիա սահմանել:

Սպորն գրված ֆունկցիան ստանում է անորոշ քանակով արգումենտներ և սրացված արգումենտների միջից գտնում է նրանք, որոնք ունեն `salary` և `per` անունները: Եթե այդպիսիք կան, ապա ֆունկցիան հաշվում է աշխատավարձի աճը՝ փրկված փոկոսով, իսկ եթե այդ թվերից որևէ մեկը բացակայում է՝ վերադարձնում է համապատասխան հաղորդագրություն: Բազմակերպով փրկված արգումենտի բոլոր արժեքները հնարավոր է պահել ցուցակի ներսում:

```
Salary_Increase<-function(...){
  Args<-list(...)
  if(is.null(Args[["salary"]]) | is.null(Args[["per"]])) x<-"Information is Missing"
  else x<-Args[["salary"]]+Args[["salary"]]*Args[["per"]]/100
  x
}
```

```
Salary_Increase(first="John", last="Tyson", salary=125000, per=25)
```

```
## [1] 156250
```

```
Salary_Increase(first="Mike", last="Mason", salary=130000, country="USA")
```

```
## [1] "Information is Missing"
```

```
Salary_Increase(first="Bill", last="Johnson", city="Seattle", company="Amazon")
```

```
## [1] "Information is Missing"
```

Բազմակերպ արգումենտները մեկ այլ կարևոր կիրառություն ևս ունի. այն օգտագործվում է նշելու համար ֆունկցիայի արգումենտները որոնք փոխանցվում են այլ ֆունկցիաների: Սա հաճախ օգտագործվում է գոյություն ունեցող ֆունկցիաներն ընդլայնելիս՝ հին ֆունկցիայի բոլոր արգումենտները չպարճենելու համար:

Որպես օրինակ դիտարկենք վերը սահմանված *Statistic()* ֆունկցիան՝ կիրառած անհայտ արժեքներ պարունակող վեկտորի վրա՝

```
Statistic <- function(func, x, n=length(x)){
  func(x[1:n])
}
```

```
Statistic(mean, c(NA,1,2,NA,3))
```

```
## [1] NA
```

Արժեքը *NA* է, քանի որ վեկտորը պարունակում է *NA* արժեքներ: Նախասահմանված *mean()* ֆունկցիան ունի արգումենտ, որի արժեքը *TRUE* դնելու դեպքում միջինը հաշվելիս վեկտորից հեռացվում են բոլոր *NA* արժեքները:

```
mean(c(NA,1,2,NA,3), na.rm=TRUE)
```

```
## [1] 2
```

Այժմ փորձենք այնպես սահմանել *Statistic()* ֆունկցիան, որ *mean()* ֆունկցիան նրան փոխանցելիս հնարավոր լինի այս արգումենտը ևս օգտագործել: Սա կարող ենք կատարել վերը նշված բազմակերպ արգումենտի միջոցով՝

```
Statistic <- function(func,x, n=length(x), ...){
  func(x[1:n],...)
}
```

```
Statistic(fun=mean, x=c(NA,1,2,NA,3), na.rm=TRUE)
```

```
## [1] 2
```

Նշելով բազմակերպ ֆունկցիայի սահմանման մեջ մենք տեղեկացնում ենք, որ *func* արգումենտին փոխանցված ֆունկցիան կարող է սրանալ նաև արգումենտներ:

### 2.4.3.1 Բինար գործողություններ (Binar Operators)

Ֆունկցիաները որոնք ունեն երկու արգումենտներ կոչվում են բինար գործողություններ: Բինար գործողությունների ամենահայտնի օրինակներն են թվաբանական գործողությունները:

```
4+5
```

```
## [1] 9
```

```
~+(4,5)
```

```
## [1] 9
```

Սահմանենք նոր բինար գործողություններ: Մտորեն սահմանված բինար գործողությունը միացնում է տրված երկու տեքստային արժեքները մեկ փոփոխականի մեջ: Բինար գործողությունը պետք է ունենա անունի հատուկ գրելաձև՝ *%name%*:

```
"%c%"<-function(x,y) paste(x,y)
```

```
"Hello" %c% "World"
```

```
## [1] "Hello World"
"Hello" %% "World" %% "of" %% "R"
## [1] "Hello World of R"
```

#### 2.4.4 Միալների վերհանումը ֆունկցիայում (Debugging)

Երբ ծրագրային իրագործումն աշխարհեցնելիս խնդիրներ են ծագում, R-ը ծրագրավորողի հետք հաղորդակցվելու համար օգտագործում է հաղորդագրությունների որոշ տեսակներ: *message*-ը ամենապարզ հաղորդագրությունն է, որի առկայությունը չի նշանակում, թե խնդիր է ծագել: Պազապես նշվում է որոշակի տեղեկատվություն կատարված հրամանների մասին: Պարզ հաղորդագրությունները հաճախ հանդիպում են նոր գրադարաններ ակտիվացնելիս, երբ կարճ ներկայացվում է թե ինչ է անհրաժեշտ այդ գրադարանների աշխարանքի համար, կամ պարզապես տեղեկացվում է գրադարանների ստեղծողների մասին:

```
message("Hello")
```

```
## Hello
```

Հաղորդագրության հաջորդ տեսակը զգուշացումն է՝ *warning*, որը ցույց է փախի, որ անսպասելի արդյունք է ստացվել, որը, սակայն, չի կասեցրել ծրագրի իրագործումը և իրագործման արդյունքը ստացվում է: Զգուշացումը ստավում է ծրագրային իրագործման արդյունքը ստացվելուց հետո և ոչ այն պահին, երբ անսպասելի արդյունքը գրացվել է: Զգուշացում ստանալու ամենահայտի օրինակներից է *if* կառուցվածքի ներսում որպես պայման գրել 1-ից ավելի երկարություն ունեցող փրամաբանական վեկտոր: Այդ դեպքում վերցվում է վեկտորի առաջին կոորդինատում գրված փրամաբանական արժեքը, կատարվում է ծրագրային ամբողջ իրագործումը և ամենավերջում փակվում է զգուշացումը:

```
warning("Hello")
```

```
## Warning: Hello
```

```
if(c(TRUE,FALSE)) print("It is TRUE!")
```

```
## Warning in if (c(TRUE, FALSE)) print("It is TRUE!"): the condition has
## length > 1 and only the first element will be used
```

```
## [1] "It is TRUE!"
```

Հաղորդակցության մյուս տեսակը սխալի տեղի ունենալու մասին հաղորդագրությունն է՝ *error*: Դրա հայտնվելը «ճակատագրական» ազդեցություն է ունենում և ծրագրի իրագործումը դադարում է, փակվում է համապատասխան հաղորդագրությունը: Միալի արտաբերման ֆունկցիան է *stop*():

```
stop("Hello")
```

Վերը նշված բոլոր հաղորդագրությունները նույն գաղափարի՝ պայմանական իրագործման (conditional execution) տարատեսակներ են: Ծրագրավորողը կարող է նախատեսել, որ իր ստեղծած ֆունկցիան բազմազան հաղորդագրություններ ուղարկի օգտագործողին՝ որոշակի պայմանների տեղի ունենալու դեպքում:

```
Sqrt<-function(x){
  if(x<0){
    warning(paste(x,"is a negative number, it is converted to a complex number"))
    x<-as.complex(x)
  }
  sqrt(x)
}
Sqrt(-1)
```

```
## Warning in Sqrt(-1): -1 is a negative number, it is converted to a complex
## number
```

```
## [1] 0+1i
Sqrt_<-function(x){
  if(x<0){
    print(paste(x,"is a negative number. Please, enter a positive number. "))
  }
  else return(sqrt(x))
}
Sqrt_(-1)
```

```
## [1] "-1 is a negative number. Please, enter a positive number."
```

```
Sqrt_(256)
```

```
## [1] 16
```

Երբ ֆունկցիան աշխատեցնելիս սխալ է տեղի ունենում շարք դժվար է բացահայտել, թե որ քայլում է տեղի ունեցել սխալը: Այդ պարճառով անհրաժեշտ է ֆունկցիայի ներսում քայլ առ քայլ կատարել հրամանները և տեսնել, թե որ քայլում է սխալը տեղի ունենում: Մխալների վերհանման (debugging) համար կարելի է օգտագործել հետևյալ հնարքը: Ենթադրենք սահմանել են ստորև բերված ֆունկցիան,

```
Average <- function(x, bycol = TRUE){
  if(!bycol) x<-t(x)
  nc <- ncol(x)
  av_values<-vector(mode="numeric", length=nc)
  for(i in seq_len(nc)){
    av_values[i]<-mean(x[,i])
  }
  av_values
}
```

և այն սխալ է տվել արգումենտների որևէ արժեքների համար կանչելիս՝

```
M1 <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12), ncol=3)
Average(M1, bycol=FALSE)
```

Որպեսզի պարզենք, թե որ քայլում է սխալը տեղի ունեցել, կարող ենք արգումենտների անուններին համապատասխան գործալ փոփոխականներ ստեղծել՝ մինչ ֆունկցիայի սահմանումը, դրանց վերագրել այն արժեքները, որոնց դեպքում ֆունկցիայում սխալ էր տեղի ունենում, և աշխատեցնել ֆունկցիայի ներսում գրված հրամանները՝ առանց ֆունկցիայի սահմանման՝

```
M1 <- matrix(c(1,2,3,4,5,6,7,8,9,10,11,12), ncol=3)
x<-M1; bycol<-FALSE
#Average <- function(x, bycol = TRUE){
  if(!bycol) x<-t(x)
  nc <- ncol(x)
  av_values<-vector(mode="numeric", length=nc)
  for(i in seq_len(nc)){
    av_values[i]<-mean(x[,i])
  }
  av_values
}
```

```
## [1] 5 6 7 8
```

```
#}
```

Այս դեպքում հրամանները չեն իրագործվի որպես մեկ ամբողջություն և հնարավորություն կլինի տեսնել այն հրամանը, որն աշխատեցնելիս սխալ է տեղի ունենում:

### 2.4.5 Արժեքների կցում (Symbol Binding)

Այս բաժնում քննարկելու ենք փոփոխականին արժեքներ կցելու սկզբունքները: Դիտարկենք նախասահմանված  $\pi$  թիվը՝

```
pi
```

```
## [1] 3.141593
```

Այժմ  $\pi$  փոփոխականին վերագրենք այլ արժեք և փպենք այն՝

```
pi<-4
pi
```

```
## [1] 4
```

Նարց է առաջանում, թե ինչու **R**-ը փպեց մեր վերագրած թիվը այլ ոչ թե նախասահմանվածը: Երբ մենք կանչում ենք փոփոխականը, ապա **R**-ը սկսում է դրա արժեքը փնտրել փարբեր միջավայրներում՝ environments, որոնք կարելի է հասկանալ որպես օբյեկտների և դրանց արժեքների ցուցակ: Չգտնելով տրված միջավայրում, այն անցնում է հաջորդ միջավայրերում փնտրելու՝ մինչ գտնելը: Երբ տրված անունով փոփոխականի արժեքը գտնվում է, որոնումը դադարում է, իսկ երբ որևէ միջավայրում այն չի գտնվում՝ վերադարձվում է սխալի տեղի ունենալու հաղորդագրությունը: Միջավայրերի հերթականությունը որոնցում կատարվում է որոնումը կարելի է տեսնել `search()` նախասահմանված ֆունկցիայի միջոցով՝

```
search()
```

```
## [1] ".GlobalEnv"      "package:stats"    "package:graphics"
## [4] "package:grDevices" "package:utils"    "package:datasets"
## [7] "package:methods" "Autoloads"       "package:base"
```

Ինչպես տեսնում ենք՝ որոնումը սկզբում կատարվում է գլոբալ միջավայրում (global environment), որտեղ պահվում են ծրագրավորողի կողմից սահմանված արժեքները, իսկ այդպիսի չգտնելու դեպքում, հերթականությամբ դիտարկվում են բոլոր նախասահմանված գրադարաններում: Երբ նոր գրադարան է կցվում, ապա այն հայտնվում է արդեն կցված գրադարաններից առաջ և սպանում առաջնահերթություն՝ որոնում կատարելիս:

```
library(HistData)
```

```
## Warning: package 'HistData' was built under R version 3.5.1
```

```
search()
```

```
## [1] ".GlobalEnv"      "package:HistData" "package:stats"
## [4] "package:graphics" "package:grDevices" "package:utils"
## [7] "package:datasets" "package:methods"  "Autoloads"
## [10] "package:base"
```

Կարևոր է հիշել, որ **R**-ը ֆունկցիաների և ոչ-ֆունկցիաների անունները փարբեր տեղերում է պահում, հետևաբար հնարավոր է միաժամանակ ունենալ, օրինակ,  $c$  անունով և՛ փոփոխական և՛ ֆունկցիա:

Փոփոխականին արժեքի կցումը ֆունկցիայի ներսում ունի իր առանձնահատկությունները, որին կանդրադառնանք հաջորդ բաժնում:

### 2.4.6 Ներդրված ֆունկցիաներ

**R**-ում հնարավոր է սահմանել ներդրված ֆունկցիաներ: Ֆունկցիայի մեջ մեկ այլ ֆունկցիայի սահմանումը ոչ բոլոր ծրագրավորման լեզուներում է հնարավոր: Այն առավելապես հարմար է վիճակագրական ֆունկցիաների հետ աշխատելիս, երբ անհրաժեշտ է ֆունկցիա սահմանել, որն ուրիշ ֆունկցիաներ է կառուցում: Երբեմն անհնաժեշտություն է առաջանում տրված երկու փոփոխականների ֆունկցիայի համար գտնել դրա նվազագույն

արժեքը ըստ փոփոխականներից որևէ մեկի՝ մյուս փոփոխականի որևէ ֆիքսված արժեքի դեպքում: Օրինակ, ֆունկցիաների փրված ընդհանրի դեպքում

$$\left\{ f(x, \theta) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-\theta)^2}{2}}, \theta, x \in \mathbb{R} \right\}$$

Եթե ուզում ենք փրված  $x = x_0$  արժեքի դեպքում սրանալ  $f_0(\theta) = f(x_0, \theta)$  կարող ենք սահմանել հետևյալ կերպ՝

```
Func <- function(x){
  function(x=x,theta) (1/sqrt(2*pi))*exp(-(x-theta)^2/2)
}
```

```
Func0<-Func(0)
class(Func0)
```

```
## [1] "function"
```

```
args(Func0)
```

```
## function (x = x, theta)
## NULL
```

#### 2.4.7 Տեսանելիության շրջանակ (Scoping Rules)

Տեսանելիության շրջանակը որոշում է այն կանոնները որոնցով ֆունկցիայի ներսում կապարվում է ազատ փոփոխականին արժեքի կցումը: **R**-ում փեղի ունի ստատիկ կցումը (lexical scoping or static scoping): Մյուս ամենաարարածված այլընտրանքը դինամիկ կցումն է (dynamic scoping):

Ֆունկցիայի ներսում կան երկու տեսակի փոփոխականներ՝ որպես արգումենտ սահմանվածները կամ հենց ֆունկցիայի ներսում սահմանվածներ և ազատ փոփոխականները որոնք արգումենտներ չեն և սահմանված չեն ֆունկցիայի ներսում, այլ սահմանված են ֆունկցիայից դուրս:

Սրափիկ կցման ժամանակ ազատ փոփոխականների արժեքները փնտրվում են այն միջավայրում, որտեղ ֆունկցիան սահմանված է: Միջավայրը (environment) փոփոխականի անուն, արժեք գույգերի բազմություն է: Յուրաքանչյուր միջավայր ունի իրեն ծնող միջավայրը (parent environment) և հնարավոր է, որ մի միջավայրը ծնի մեկից ավելի միջավայրներ: Միակ միջավայրն առանց իրեն ծնող միջավայրի դափարկ միջավայրն է (empty environment), իսկ ֆունկցիան իրեն ծնող միջավայրի հետ միասին կոչվում է լրացում կամ ֆունկցիայի լրացում (closure or function closure): Տրված անունով ազատ փոփոխականի արժեքը գտնելու համար փնտրումը նախ կարարվում է այն միջավայրում, որտեղ ֆունկցիան սահմանված է: Այդտեղ չգտնելու դեպքում որոնումը շարունակվում է նշված միջավայրը ծնող միջավայրում և այդպես շարունակ մինչև գլոբալ միջավայրը: Եթե այս ընթացքում արժեքը չի գտնվում, որոնումը փոխում է ուղղությունը և սկսում ներքև իջնել միջավայրների մեջ փնտրելով մինչ կհասնի դափարկ միջավայրին: Եթե այս ընթացքում ևս փոփոխականի արժեքը չի գտնվում փեղի է ունենում սխալ:

##### 2.4.7.1 Օրինակներ

1. Դիփարկենք հետևյալ ֆունկցիաները՝

```
fun1 <- function(x) x+y

fun2 <- function() {
  y<-20
  function(x) x+y
}
```

Առաջինը սովորական ֆունկցիա է որում  $y$ -ն ազատ փոփոխական է, իսկ երկրորդը ֆունկցիա է, որը ստեղծում է այլ ֆունկցիա: Երկրորդի միջոցով ստեղծենք  $fun3()$  ֆունկցիան՝

```
fun3<-fun2()
```

Դիտարկենք  $fun1()$  և  $fun3()$  ֆունկցիաների կառուցվածքները՝

```
fun1
```

```
## function(x) x+y
```

```
fun3
```

```
## function(x) x+y
```

```
## <environment: 0x000000001947d8a0>
```

Ինչպես տեսնում ենք՝ երկու ֆունկցիաներն էլ ունեն նույն կառուցվածքը և կապարում են նույն գործողությունը, միակ փարբերությամբ, որ երկրորդ ֆունկցիայի սահմանման հետ փրված է նաև այն միջավայրը որում այն սահմանված է: Սա արվում է այն դեպքում, երբ ֆունկցիան սահմանված չէ գլոբալ միջավայրում, այլ որևէ ֆունկցիայի ներսում:

```
y<-10
```

```
fun1(5)
```

```
## [1] 15
```

```
fun3(5)
```

```
## [1] 25
```

Ինչպես տեսնում ենք՝ երկու ֆունկցիաներն ունեն նույն կառուցվածքը, բայց փարբեր արժեքներ վերադարձրեցին արգումենտի նույն արժեքի դեպքում: Պարզապես ֆունկցիայի ներսում առկա անկախ փոփոխականին արժեք վերագրելու սկզբունքի մեջ է: Երկրորդ ֆունկցիան սահմանված է մեկ այլ ֆունկցիայի ներսում, հետևաբար ազատ փոփոխականի արժեքը փնտրվում է, հենց այդ՝ սկզբնական ֆունկցիայի մեջ, այլ ոչ թե գլոբալ միջավայրում:

```
environment(fun1)
```

```
## <environment: R_GlobalEnv>
```

```
environment(fun3)
```

```
## <environment: 0x000000001947d8a0>
```

```
get("y",environment(fun1))
```

```
## [1] 10
```

```
get("y", environment(fun3))
```

```
## [1] 20
```

- Տեսնենք թե ինչ առավելություն է փալիս սրափիկ տեսանելիության հարկությունը  $\mathbb{R}$ -ում: Դիտարկենք  $\mathcal{N}(\theta, \sigma^2)$  նորմալ բաշխման ճշմարտանմանության ֆունկցիան՝ փրված (հայտնի)  $\sigma > 0$  արժեքի դեպքում: Այդ բաշխման խտության ֆունկցիան է՝

$$\left\{ f(x, \theta, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\theta)^2}{2\sigma^2}}, \theta, x \in \mathbb{R} \right\}, \sigma > 0.$$

Իսկ ճշմարտանմանության ֆունկցիան փրված  $X^n = (X_1, \dots, X_n)$  նմուշի համար կլինի՝

$$L(X^n, \theta) = \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} e^{-\frac{1}{2\sigma^2} \sum_{i=1}^n (X_i - \theta)^2} = \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} e^{-\frac{n}{2\sigma^2} \left( \frac{1}{n} \sum_{i=1}^n X_i^2 - \frac{2\theta}{n} \sum_{i=1}^n X_i + \theta^2 \right)}, \theta, x \in \mathbb{R}, \sigma > 0.$$

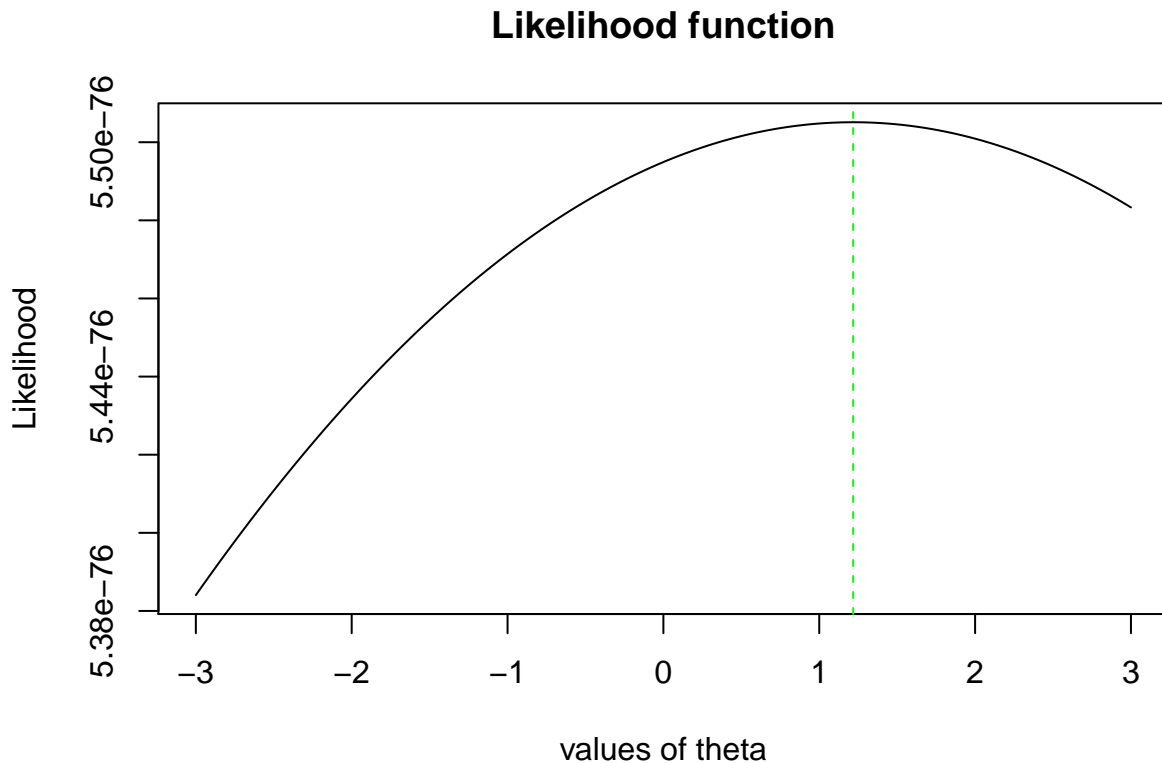


Տրված  $X^n$  ընտրանքի և  $\sigma$  դրական թվի համար պետք է գրել այս ֆունկցիայի առավելագույն արժեքը, հետևաբար նախ գրենք ֆունկցիա, որ կկառուցի այս ճշմարտանմանության ֆունկցիան:

```
Like <- function(X,sigma){
  n<-length(X)
  Mean_X <- mean(X)
  Mean_X2 <- mean(X^2)

  function(x=Mean_X,y=Mean_X2,size=n,sd=sigma,theta){
    (2*pi*sd^2)^(-size/2)*exp(-1/(2*sd^2*size)*(y-2*theta*x+theta^2))
  }
}
set.seed(1)
X<-rnorm(100,1,2)
Opt<-Like(X,sigma=2)
class(Opt)
```

```
## [1] "function"
```



Վիճակագրությունից գիտենք, որ ճշմարտանմանության ֆունկցիան իր առավելագույն արժեքին հասնում է  $\hat{\theta}_n = \frac{1}{n} \sum_{i=1}^n X_i$  կերպում՝

```
mean(X)
```

```
## [1] 1.217775
```

Ինչպես տեսնում ենք `Like()` ֆունկցիայի ներսում սահմանված ֆունկցիան ունի ընդամենը մեկ ազատ փոփոխական՝  $\theta$ , որին արժեք է կցվում գործալ միջավայրից (այն իրականում առկա չէ գործալ միջավայրում և որոնումը շարունակվում է նախասահմանված գրադարաններում՝ մինչ `base` գրադարանից արժեքի կցումը): [R](#)

հաշվողական միջավայրը, ինչպես նշել են, առաջացել է  $S$  հաշվողական լեզվից և հիմնական փարբերությունն այս լեզուների մեջ հենց փեսանելիության շրջանակն է:  $S$  լեզվում ազար փոփոխականների վերագրումը կարարվում է միայն գորբալ միջավայրում, անգամ եթե րրված ֆունկցիան ներդրված է մեկ այլ ֆունկցիայում: Ներևաբար վերը գրված ծրագիրը կաշխարի ինչպես  $S$ -ում այնպես և  $R$ -ում: Բայց ի փարբերություն  $S$ -ի  $R$ -ում ազար փոփոխականների վերագրումը կարարվում է ֆունկցիայի սահմանման միջավայրում, հերևաբար վերը նշված ծրագիրը կարելի է պարգեցնել`

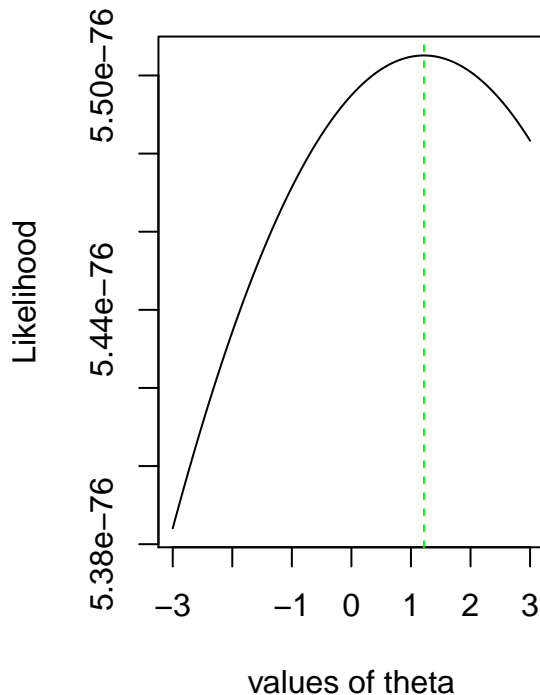
```
Like0 <- function(X,sigma){
  n<-length(X)
  Mean_X <- mean(X)
  Mean_X2 <- mean(X^2)

  function(theta){
    (2*pi*sigma^2)^(-n/2)*exp(-1/(2*sigma^2*n)*(Mean_X2-2*theta*Mean_X+theta^2))
  }
}
set.seed(1)
X<-rnorm(100,1,2)
Opt0<-Like0(X,sigma=2)
class(Opt0)

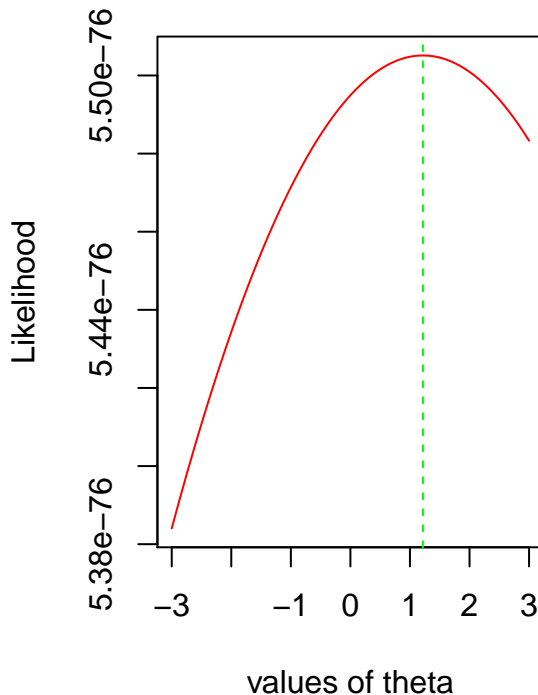
## [1] "function"
args(Opt0)

## function (theta)
## NULL
```

Definition 1



Definition 2



Սրացվեց, որ ֆունկցիայի ներսում սահմանված ֆունկցիան իր ազապ փոփոխականներին արժեքներ կցելիս կարող է օգտագործել ինչպես սկզբնական ֆունկցիայի արգումենտները, այնպես էլ սկզբնական ֆունկցիայի ներսում սահմանված լոկալ փոփոխականները: Նախորդ դեպքում այս երկու տեսակի արժեքները պետք է փոխանցվեն որպես երկրորդ ֆունկցիայի փոփոխականներ, որպեսզի երկրորդ ֆունկցիան կարողանա դրանք օգտագործել:

```
int<-seq(-3,3,0.001)
X<-rnorm(100,1,2)
f1<-Like0(X,sigma=2)
optimize(f1,int, maximum = TRUE) #optimize by default minimizes
```

```
## $maximum
## [1] 0.9244036
##
## $objective
## [1] 5.502102e-76
```

```
mean(X)
```

```
## [1] 0.9243838
```

```
X<-rnorm(100,1,2)
f2<-Like0(X,sigma=2)
optimize(f2,int,maximum = TRUE)
```

```
## $maximum
## [1] 1.059357
##
## $objective
## [1] 5.497959e-76
```

```
mean(X)
```

```
## [1] 1.059347
```

Ինչպես տեսնում ենք՝ փարբեր  $X$  նմուշների միջոցով կառուցվում են փարբեր՝  $f_1, f_2$  ֆունկցիաներ և յուրաքանչյուրն օգտագործում է այն նմուշը, որի միջոցով այն կառուցվել է: Սա տեղի է ունենում քանի որ յուրաքանչյուր ֆունկցիա իր մեջ պահում է նաև այն միջավայրը, որում այն կառուցվել է: Սա կոչվում է ֆունկցիայի փակում՝ function closure:

3. Նաջորդ օրինակը ցույց է տալիս, թե ինչով է սրափիկ կցումը փարբերվում դինամիկ կցումից:

```
y<-10

f<-function(x){
  y<-2
  y^2+g(x)
}

g <- function (x) {
  x*y
}
```

Կախված նրանից թե կցման ինչ փարբերակ է օգտագործում հաշվողական լեզուն,  $f(3)$  ֆունկցիայի կանչը կրա փարբեր արժեքներ:

```
f(3)
```

```
## [1] 34
```

Սրափիկ կցման ժամանակ (հերևաբար նաև  $\mathbb{R}$ -ում),  $g()$  ֆունկցիայի ներսում առկա  $y$  ազատ փոփոխականի արժեքը կցվում է  $g()$  ֆունկցիայի սահմանման ժամանակ և, հերևաբար, դրա արժեքը փնտրվում է միջավայրում, որտեղ  $g()$  ֆունկցիան սահմանվել է, այսինքն՝ այս դեպքում գլոբալ միջավայրում, հերևաբար  $y = 10$ : Դիմամիկ կցման ժամանակ  $y$  արժեքի վերագրումը կկարարվեր այն ժամանակ, երբ  $g()$  ֆունկցիան կանչվեր, այսինքն՝  $f()$  ֆունկցիայի ներսում, հերևաբար  $y$ -ի արժեքը կփնտրվեր  $g()$ -ի կանչման միջավայրում, որը կոչվում է նաև  $g()$ -ի ծնող կառուցվածք (parent frame):

4. Երբ ֆունկցիան սահմանված է գլոբալ միջավայրում և կանչվում է գլոբալ միջավայրից, ապա ֆունկցիայի կանչման և սահմանման միջավայրերը նույնն են և կարող է թվալ, որ րեղի է ունենում դիմամիկ կցում՝

```
g <- function(x){
  a <- 3
  x+a+y
}

# g(2) will give an error because "y" is not defined

y <- 3
g(2)
```

```
## [1] 8
```

Թվում է՝  $g()$  ֆունկցիայի ներսում առկա  $y$  ազատ փոփոխականին արժեքի կցումը կարարվեց ոչ թե  $g()$  ֆունկցիայի սահմանման ժամանակ, քանի որ այդ ժամանակ դեռևս  $y$ -ը սահմանված չէր, այլ  $g()$ -ի կանչման ժամանակ: Իրականում այսպեղ ևս րեղի է ունենում սրափիկ կցում, պարզապես ֆունկցիայի սահմանման և կանչման միջավայրների նույնը լինելու պարզառով նման րպավորություն է սրեղծվում:

Սրափիկ կցման հերևանքներից է, որ բոլոր  $\mathbb{R}$ -ում սահմանված բոլոր օբյեկտները պերք է պահվեն հիշողության մեջ, ինչպես նաև բոլոր ֆունկցիաները պերք է պահեն ցուցիչ միջավայրի վրա որտեղ իրենք սահմանվել են:

#### 2.4.8 Վերագրման գործողություններ

$\mathbb{R}$ -ում կան երեք վերագրման գործողություններ՝  $<-$ ,  $=$ ,  $<-<$ : Դրանց միջև րարբերությունը այդ վերագրումների րեսանելիության շրջանակն է, որտեղ են դրանց միջոցով վերագրում կարարված փոփոխականները հասանելի: Դիարարկենք հերևյալ օրինակները՝

1. Քանի՞ փոփոխական կսրեղծի հերևյալ հրամանը՝

```
rm(list=ls())
x<-c(a=1,b=3)
```

##### Պարասխան

Ընդամենը մեկ փոփոխական՝  $x$ -ը: Մյուս վերագրման գործողությունն օգրագործվում է  $c()$  ֆունկցիայի արգումենտում արժեքներին անուններ րալու համար: Այսինքն՝  $a$  և  $b$  փոփոխականները գոյություն ունեն միայն  $x$  փոփոխականի ներսում:

```
ls()

## [1] "x"
x

## a b
## 1 3
```

2. Քանի՞ փոփոխական կսրեղծի հերևյալ հրամանը՝

```
rm(list=ls())
x<-c(a<-1, b<-3)
```

### Պատասխան

Այն կսպեղծի երեք փոփոխականներ՝ գորբալ միջավայրում: Թեև  $a$  և  $b$  փոփոխականներին արժեք է վերագրվում  $c()$  ֆունկցիայի արգումենտում, բայց նշված վերագրման արտակի պարճառով վերագրումը կարարվում է գորբալ միջավայրում:

```
ls()

## [1] "a" "b" "x"
x
```

```
## [1] 1 3
```

3. Ի՞նչ կվերադարձնի հետևյալ հրամանը՝

```
rm(list=ls())
mean(z=1:4)
```

### Պատասխան

Այսպեղ արելի է ունենալու սխալ, որովհետև  $mean()$  ֆունկցիան չունի  $z$  անունով արգումենտ, նաև ֆունկցիայի արգումենտում վեկտորի  $z = 1 : 4$  սահմանումը արեսանելի չէ գորբալ միջավայրում, իսկ  $mean()$  ֆունկցիան արժեքները փնարում է գորբալ միջավայրում:

```
## <simpleError in mean.default(z = 1:4): argument "x" is missing, with no default>
```

4. Նախորդ օրինակը ձևափոխելով հետևյալ կերպ, հաշվարկը արելի կունենա

```
rm(list=ls())
mean(z<-1:4)
```

```
## [1] 2.5
```

5. Վերագրման երրորդ օրինակը  $<- <-$  թույլ է րալիս ֆունկցիայի ներսում կարարել վերագրում, որը արեսանելի կլինի այն միջավայրում, որպեղ ֆունկցիան սրեղծվել է:

```
rm(list=ls())
x<-10
change<-function() x<<- 12
change()
x
```

```
## [1] 12
```

Վերագրման այս արեսակը կարող է օգրագործվել, օրինակ, հաշվելու համար թե րրված ֆունկցիան քանի անգամ է կանչվել:

```
rm(list=ls())
i<-0

Mean<-function(x,...){
  i<<-i+1
  mean(x,...)
}
```

```
Mean(1:10)
```

```
## [1] 5.5
```

```
Mean(seq(0,1,0.1))
```

```
## [1] 0.5
```

```
Mean(c(4,-5,12,-6))
```

```
## [1] 1.25
```

```
i
```

```
## [1] 3
```

6. Քանի որ **R**-ում րեղի ունի սպարիկ կցում, ապա `<-` վերագրումը ֆունկցիայի ներսում կատարելիս այն կփոխի ոչ թե ֆունկցիայի կանչման միջավայրում փոփոխականի արժեքը, այլ ֆունկցիայի սահմանման միջավայրում: Տե՛ս հետևյալ օրինակը՝

```
rm(list=ls())
```

```
change<-function() {x<-12;print("The function `change` is executed")}
```

```
x<-10
```

```
f<-function(){
  x<-5
  change()
  x
}
```

```
f()
```

```
## [1] "The function `change` is executed"
```

```
## [1] 5
```

```
x
```

```
## [1] 12
```

Փոխարենը՝

```
rm(list=ls())
```

```
x<-10
```

```
f<-function(){
  x<-5
  change<-function() {x<-12;print("The function `change` is executed")}
  change()
  x
}
```

```
f()
```

```
## [1] "The function `change` is executed"
```

```
## [1] 12
```

```
x
```

```
## [1] 10
```

## 2.4.9 Խնդիրներ

1. Գրել ֆունկցիա, որը որպես արգումենտներ ստանում է երկու ուղղանկյունաձև փվյալային կառուցվածք, որոնցից առաջինը պարունակում է չափումներ, իսկ երկրորդը՝ այդ չափումների համար նախատեսված նորմալության միջակայքերը: Պետք է ստուգել, թե առաջին ուղղանկյուն փվյալներում յուրաքանչյուր չափում գրնվում է իր նորմալության միջակայքում, դրանից վերև, թե՛ ներքև և ստեղծել այդ արդյունքները պարունակող սյուններ: Որպես առաջին ուղղանկյուն կառուցվածք կարող ենք դիֆարկել *airquality* ուղղանկյուն կառուցվածքը, իսկ երկրորդի օրինակ կարող է լինել՝

```
NR <- data.frame(Variable=c("Wind", "Temp"), Low=c(7,70), High=c(10,80))
NR
```

```
## Variable Low High
## 1 Wind 7 10
## 2 Temp 70 80
```

**Լուծում**

```
Add_Col <- function(dat, NR){
  x<-NR[,"Variable"]
  for(i in x){
    dat[,paste0(i,"FL")]<-ifelse(dat[,i]<NR[NR$Variable==i,"Low"],"Below",
                                ifelse(dat[,i]<=NR[NR$Variable==i,"High"],
                                        "Normal", "High"))
  }
  dat
}
```

```
head(Add_Col(airquality,NR))
```

```
## Ozone Solar.R Wind Temp Month Day WindFL TempFL
## 1 41 190 7.4 67 5 1 Normal Below
## 2 36 118 8.0 72 5 2 Normal Normal
## 3 12 149 12.6 74 5 3 High Normal
## 4 18 313 11.5 62 5 4 High Below
## 5 NA NA 14.3 56 5 5 High Below
## 6 28 NA 14.9 66 5 6 High Below
```

```
table(Add_Col(airquality,NR)$WindFL)
```

```
##
## Below High Normal
## 33 72 48
```

```
table(Add_Col(airquality,NR)$TempFL)
```

```
##
## Below High Normal
## 32 68 53
```

```
table(Add_Col(airquality,NR)$WindFL, Add_Col(airquality,NR)$TempFL)
```

```
##
## Below High Normal
## Below 0 22 11
## High 22 20 30
## Normal 10 26 12
```

Կարող ենք նորմալության միջակայքեր պարունակող մեկ այլ ուղղանկյուն փվյալներ վերցնել՝

```
NR2 <- data.frame(Variable=c("Temp", "Ozone", "Wind"), Low=c(70,50, 7),
                    High=c(80, 60, 10))
```

```
NR2
```

```
## Variable Low High
## 1 Temp 70 80
## 2 Ozone 50 60
## 3 Wind 7 10
```

Կիրառելով նույն ֆունկցիան, կարանանք՝

```
head(Add_Col(airquality, NR2))
```

```
## Ozone Solar.R Wind Temp Month Day TempFL OzoneFL WindFL
## 1 41 190 7.4 67 5 1 Below Below Normal
## 2 36 118 8.0 72 5 2 Normal Below Normal
## 3 12 149 12.6 74 5 3 Normal Below High
## 4 18 313 11.5 62 5 4 Below Below High
## 5 NA NA 14.3 56 5 5 Below <NA> High
## 6 28 NA 14.9 66 5 6 Below Below High
```

```
table(Add_Col(airquality, NR2)$WindFL)
```

```
##
## Below High Normal
## 33 72 48
```

```
table(Add_Col(airquality, NR2)$TempFL)
```

```
##
## Below High Normal
## 32 68 53
```

```
table(Add_Col(airquality, NR2)$OzoneFL)
```

```
##
## Below High Normal
## 81 31 4
```

#### 2.4.10 Ամփոփում

- Arguments
- Anonymous functions
- Lazy Evaluation
- Symbol Binding
- Environments, Global Environment, Parent Environment
- Static or Lexical Scoping, Dynamic Scoping
- Parent Frame
- Function Closure
- Debugging
- Conditional Execution
- Message, Warning, Error



## Functions

- `args()`
- `search()`
- `environment()`
- `get()`
- `optimize()`
- `message()`, `warning()`, `stop()`

## 2.5 Split - Apply - Combine


Այս բաժնում ուսումնասիրելու ենք շրջապտույտի ֆունկցիաները՝ Loop Functions, որոնք օգտագործվում են կիրառելու համար բաժանել-կիրառել-միավորել մարտավարությունը՝ (Split-Apply-Combine): Այս մարտավարության նպատակն է բաժանել տվյալները ավելի փոքր մասերի, կիրառել տրված ֆունկցիան մասերի վրա և միավորել սրացված արդյունքները: Ստորև բերված աղյուսակն ամփոփ ներկայացնում է շրջապտույտի ֆունկցիաները: Չախ կողմում գրված տողերը ցույց են տալիս օբյեկտի տեսակը, որը փոխանցվում է ֆունկցիային, իսկ սյուները ցույց են տալիս ֆունկցիայի կողմից վերադարձվող օբյեկտի տեսակը: Գունավորում ունեցող ֆունկցիաները -ում նախասահմանված ֆունկցիաներն են, իսկ սպիտակ ֆունկցիաները Նադլեյ Ուիքհեմի (Hadley Wickham) կողմից ստեղծված `plyr` գրադարանից են:

Figure 2.2: Շրջապտույտի ֆունկցիաներ

	array	data frame	list	nothing
array	<code>apply</code>	<code>adply</code>	<code>alply</code>	<code>a_ply</code>
data frame	<code>dapply</code>	<code>aggregate</code>	<code>by</code>	<code>d_ply</code>
list	<code>sapply</code>	<code>ldply</code>	<code>lapply</code>	<code>l_ply</code>

### 2.5.1 `lapply()`, `sapply()`

Ենթադրենք ունենք ցուցակ, որի մեջ գրված են երեք մատրիցներ և մենք ուզում ենք հաշվել դրանց յուրաքանչյուրի որոշիչը:

```
l<-list(M1=matrix(c(2,1,3,3),nrow=2),M2=matrix(c(12,31,44,2),nrow=2),
        M3=matrix(c(18,87,0,-2),nrow=2))
```

`det()` ֆունկցիային կարելի է որպես արգումենտ տալ միայն մատրից: Որպեսզի կարողանանք այն կիրառել ցուցակի բոլոր ենթամատրիցների վրա անհրաժեշտ է շրջապտույտի `for` հրամանով հերթով դուրս հանել ցուցակի բոլոր ենթամատրիցները:

```
deter<-c()

for(x in l){
  deter <- c(deter, det(x))
}
```

```
deter
```

```
## [1]      3 -1340   -36
```

Այս ամբողջն ավելի հեշտ կլիներ եթե կարողանայինք [R](#)-ին ուղղորդել, որպեսզի այն կիրառեր նույն `det()` ցուցակի բոլոր ենթամասորիցների համար: Նենց սրա համար օգտագործում ենք `lapply()` ֆունկցիան, որն ունի երեք արգումենտ: Նրան նախ պետք է փալ ցուցակ, որի վրա պետք է ֆունկցիա կիրառել, այնուհետև ֆունկցիան, որն ուզում ենք կիրառել և բազամեկ արգումենտում կարող ենք նշել բոլոր այն արգումենտները, որոնք ուզում ենք փոխանցել նախորդ ֆունկցիային: `lapply()` ֆունկցիան միշտ ցուցակ է վերադարձնում:

```
lapply(1, deter)
```

```
## $M1
## [1] 3
##
## $M2
## [1] -1340
##
## $M3
## [1] -36
```

Ինչպես երևում է՝ բոլոր մասորիցների անունները պահպանվել են և յուրաքանչյուրի համար հաշվվել է դրա որոշիչը:

Այժմ ենթադրենք ունենք ցուցակ, որում գրված են երեք վեկտորներ և անհրաժեշտ է հաշվել յուրաքանչյուրի միջին թվաբանականը:

```
l <- list(a=c(12,33,44,55), b=c(56,-16,NA),
         c=c(85,654,-567,-10))
lapply(1, mean)
```

```
## $a
## [1] 36
##
## $b
## [1] NA
##
## $c
## [1] 40.5
```

Երկրորդ վեկտորի միջին թվաբանականը `NA` է քանի որ այդ վեկտորը պարունակում է `NA` արժեք: Ինչպես գիտենք՝ `mean()` ֆունկցիան ունի արգումենտ, որի միջոցով հնարավոր է վեկտորից դուրս հանել `NA` արժեքները՝ մինչ միջին թվաբանականը հաշվելը, ապա հաշվել միջին թվաբանականը: Բազմակերպ արգումենտի միջոցով `mean()` ֆունկցիայի այս արգումենտը կարող ենք փոխանցել `lapply()` ֆունկցիային:

```
lapply(1, mean, na.rm=TRUE)
```

```
## $a
## [1] 36
##
## $b
## [1] 20
##
## $c
## [1] 40.5
```

`lapply()` ֆունկցիան, որպես արգումենտ ֆունկցիա է ընդունում, հերևաբան, ինչպես և նմանափայ այլ դեպքերում հնարավոր է որպես արգումենտ փոխանցել անանուն ֆունկցիա: Դիտարկենք մասորիցներ

պարունակող նախկին ցուցակը և փորձենք դուրս հանել այդ մարրիցներից յուրաքանչյուրի առաջին փողը:

```
l<-list(M1=matrix(c(2,1,3,3),nrow=2),M2=matrix(c(12,31,44,2),nrow=2),
        M3=matrix(c(18,87,0,-2),nrow=2))
```

```
l
## $M1
##      [,1] [,2]
## [1,]    2    3
## [2,]    1    3
##
## $M2
##      [,1] [,2]
## [1,]   12   44
## [2,]   31    2
##
## $M3
##      [,1] [,2]
## [1,]   18    0
## [2,]   87   -2
```

```
lapply(l, function(x) x[1,])
```

```
## $M1
## [1] 2 3
##
## $M2
## [1] 12 44
##
## $M3
## [1] 18 0
```

*sapply()* ֆունկցիան կապարում է նույն գործողությունները ինչ *lapply()* ֆունկցիան, պարզապես հնարավորության դեպքում վերադարձվող ցուցակը պարզեցվում է (*s=simplify*) և վերադարձվում է համասեռ վեկտոր (երբ վերադարձվող ցուցակի յուրաքանչյուր կորդինատը մեկ երկարությամբ համասեռ վեկտոր է), մարրից (եթե վերադարձվող ցուցակի յուրաքանչյուր կորդինատը մեկից մեծ և մյուսներին հավասար երկարությամբ համասեռ վեկտոր է), իսկ երբ հնարավոր չէ պարզեցնել, վերադարձվում է ցուցակ:

```
l<-list(M1=matrix(c(2,1,3,3),nrow=2),M2=matrix(c(12,31,44,2),nrow=2),
        M3=matrix(c(18,87,0,-2),nrow=2))
```

```
sapply(l,det)
```

```
##      M1      M2      M3
##      3 -1340  -36
```

```
l <- list(a=c(12,33,44,55), b=c(56,-16,NA),
         c=c(85,654,-567,-10))
```

```
sapply(l, mean, na.rm=TRUE)
```

```
##      a      b      c
## 36.0 20.0 40.5
```

```
l<-list(M1=matrix(c(2,1,3,3),nrow=2),M2=matrix(c(12,31,44,2),nrow=2),
        M3=matrix(c(18,87,0,-2),nrow=2))
```

```
sapply(l, function(x) x[1,])
```

```
##      M1 M2 M3
## [1,]  2 12 18
```

```
## [2,] 3 44 0
```

### 2.5.2 apply()

`apply()` ֆունկցիան կիրառվում է զանգվածների, այդ թվում մարրիցների վրա: Այս ֆունկցիայի առաջին արգումենտը զանգվածն է, որի վրա ուզում ենք ֆունկցիա կիրառել, երկրորդը՝ չափողականությունն է, որն օգտագործվելու է զանգվածի հետ աշխատելիս, իսկ երրորդը՝ ֆունկցիան է, որը պետք է կիրառել: Բազմակերպ արգումենտը ևս առկա է: Որպես օրինակ դիտարկենք հետևյալ մարրիցը՝

```
M <- matrix(1:20,ncol=4)
M
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  1   6  11  16
## [2,]  2   7  12  17
## [3,]  3   8  13  18
## [4,]  4   9  14  19
## [5,]  5  10  15  20
```

Օգտվելով նախորդ ֆունկցիայից՝ կարող ենք հաշվել այս մարրիցի սյուների կամ սյունների միջինները, հիշելով, որ սյուներն առաջին չափողականությունն են, սյունները՝ երկրորդ:

```
apply(M, 1, mean)
```

```
## [1] 8.5 9.5 10.5 11.5 12.5
```

```
apply(M, 2, mean)
```

```
## [1] 3 8 13 18
```

`apply()` ֆունկցիայի վերը նշված գործածությունն այնքան հաճախակի է, որ գոյություն ունեն առաձին ֆունկցիաներ, որոնք կատարում են նշված գործողությունները՝ էլ ավելի հեշտացնելով այս ֆունկցիաների կիրառումը:

```
rowSums(x) = apply(x,1,sum)
rowMeans(x) = apply(x,1,mean)
colSums(x) = apply(x,2,sum)
colMeans(x) = apply(x,2,mean)
```

Բացի ավելի կարճ գրառում ունենալը, այս ֆունկցիաները շատ ավելի արագ կիրառվեն հաշվարկը, քան `apply()` ֆունկցիայի միջոցով նույնն անելը: Եթե կիրառելու ենք այլ ֆունկցիաներ, ապա կարող ենք փոխանցել այդ ֆունկցիաների արգումենտները ևս՝

```
apply(x, 1, quantile, probs=c(0.25, 0.75))
```

```
##      [,1] [,2]
## 25%  4.5 20.25
## 75% 13.5 64.75
```

Տեսնում ենք, որ արդյունքները միավորվել են և ներկայացվում են մարրիցի տեսքով:

### 2.5.3 mapply()

Նախորդ ֆունկցիաները փոխանցում են օբյեկտի ենթաօբյեկտները հաջորդաբար փոխանցում էին փոխանցում ֆունկցիային՝ որպես արգումենտ: Այժմ ենթադրենք, որ ունենք մի քանի օբյեկտներ, որոնց ենթաօբյեկտները հաջորդաբար փոխանցվում են փոխանցում ֆունկցիայի փարբեր արգումենտներին: Ֆունկցիայի միաժամանակյա կանչումը փարբեր փոփոխականներին արժեքներ փոխանցելու կատարվում է `mapply()` ֆունկցիայի միջոցով: Դիտարկենք հետևյալ օրինակը: Ենթադրենք փոխանցում են հետևյալ ուղղանկյուն փոխանցումը՝

```
M <- airquality[,1:4]
```

Նաշվենք ամեն սյան քանորդիչը, ընդ որում՝ առաջինի համար հաշվում ենք 0.1 քանորդիչը, երկրորդի համար՝ 0.25, երրորդի՝ 0.5 և չորրորդի՝ 0.75:

```
mapply(quantile, M, c(0.1,0.25,0.5,0.75), na.rm=T)
```

```
##      Ozone.10% Solar.R.25%      Wind.50%      Temp.75%
##      11.00      115.75      9.70      85.00
```

*mapply()* ֆունկցիայի առաջին արգումենտը ֆունկցիան է, որ ուզում ենք կիրառել, երկրորդը արգումենտը փոխանցվելու է որպես այդ ֆունկցիայի առաջին արգումենտ, երկրորդը՝ որպես երկրորդ արգումենտ, բացի այդ, կիրառվող ֆունկցիայի ևս մեկ արգումենտի լռելյալ արժեքը փոխվել է՝ *na.rm = T*:

#### 2.5.4 vapply(), tapply()

*vapply()* կարարում է նույն գործառույթները ինչ *sapply()* ֆունկցիան փարբերությամբ, որ վերջինը փորձում է հնարավորինս պարզեցնել վերջնական արդյունքները, իսկ առաջինը թույլ է տալիս օգտագործողին նշել տեսակը, որով, եթե հնարավոր է, ուզում ենք սրանալ արդյունքները: Եթե նշված տեսակով հնարավոր չէ արդյունքները ներկայացնել, տեղի է ունենում սխալ: Սա թույլ է տալիս խոսափել անակնկալ տեսքով տվյալները սրանալուց, որը հեղափոխում կարող է սխալ արդյունքի բերել:

```
vapply(airquality, class, character(1))
```

```
##      Ozone      Solar.R      Wind      Temp      Month      Day
## "integer" "integer" "numeric" "integer" "integer" "integer"
```

Նաջորդ ֆունկցիան հենց իրագործում է բաժանել-կիրառել-միավորել մարտավարությունը: Ենթադրենք պետք է տվյալները բաժանել մասերի, կիրառել մասերի վրա տրված ֆունկցիան և միավորել սրացված արդյունքները: Ֆունկցիան կոչվում է *tapply()*, որտեղ *t* նշանակում է *table*: Դիտարկենք նախասահմանված *UCBAdmissions* ուղղանկյունաձև տվյալները, որը պարունակում է Բերկլիի համալսարանի վեց ամենամեծ դեպարտամենտներում ընդունված ուսանողների քանակները 1973 թվականին:

```
UCBAdmissions <- as.data.frame(UCBAdmissions)
dim(UCBAdmissions)
```

```
## [1] 24 4
```

```
head(UCBAdmissions)
```

```
##      Admit Gender Dept Freq
## 1 Admitted  Male   A  512
## 2 Rejected  Male   A  313
## 3 Admitted Female  A   89
## 4 Rejected Female  A   19
## 5 Admitted  Male   B  353
## 6 Rejected  Male   B  207
```

Նաշվենք ընդունված բոլոր ուսանողների մեջ աղջիկների և տղաների քանակները՝

```
tapply(UCBAdmissions$Freq, UCBAdmissions$Gender, sum)
```

```
##      Male Female
##      2691  1835
```

Ինչպես տեսնում ենք՝ ֆունկցիայի առաջին արգումենտը փոփոխականն է, որն ուզում ենք մասերի բաժանել, երկրորդը՝ փոփոխականը, որի միջոցով ուզում ենք տրոհել առաջին փոփոխականը, իսկ երրորդ արգումենտը ֆունկցիան է, որն ուզում ենք կիրառել տրոհված փոփոխականի վրա:

Կարող ենք անգամ խմբավորման համար երկու փոփոխականներ վերցնել: Այդ դեպքում փոփոխականները պետք է ընդգրկել ցուցակի մեջ և փոխանցել որպես երկրորդ արգումենտ՝

```
tapply(UCBAdmissions$Freq, list(UCBAdmissions$Gender,UCBAdmissions$Admit), sum)
```

```
##           Admitted Rejected
## Male           1198     1493
## Female           557     1278
```

Սա փախի է յուրաքանչյուր սեռի ընդունված և չընդունված ուսանողների ընդհանուր քանակը:

### 2.5.5 split()

*split()* ֆունկցիան բաժանում է արված օբյեկտը մասերի՝ օգտագործելով որպես երկրորդ արգումենտ արված ֆակտորը կամ ֆակտորների ցուցակը: Այս ֆունկցիան միշտ ցուցակ է վերադարձնում, հետևաբար արդյունքը կարող ենք փոխանցել *sapply()* կամ *lapply()* ֆունկցիաներին: *split()* և *lapply()* ֆունկցիաների միաժամանակյա կիրառումը կրա նույն արդյունքը ինչ *tapply()* ֆունկցիայի կիրառումը փարբերությամբ, որ վերջինս փրոհում է վեկտորը, հետևաբար որպես արգումենտ փոխանցվող ֆունկցիան կարող է կիրառվել միայն վեկտորի վրա, իսկ *split()* ֆունկցիան կարող է փրոհել ոչ միայն վեկտոր, հետևաբար *lapply()* ֆունկցիային որպես արգումենտ փոխանցված ֆունկցիան կարող է կիրառվել նաև ոչ վեկտորի վրա: Որպես օրինակ նորից դիտարկենք *airquality* ուղղանկյունաձև փվյալները: Ենթադրենք ուզում ենք հաշվել Օզոնի, Ճառագայթման, Քամու և Ջերմաստիճանի միջին արժեքները՝ ըստ ամսիների:

```
sapply(split(airquality[,c("Ozone", "Solar.R", "Wind", "Temp")],
              airquality$Month), colMeans, na.rm=TRUE)
```

```
##           5           6           7           8           9
## Ozone    23.61538  29.44444  59.115385  59.961538  31.44828
## Solar.R  181.29630 190.16667 216.483871 171.857143 167.43333
## Wind     11.62258  10.26667   8.941935   8.793548  10.18000
## Temp     65.54839  79.10000  83.903226  83.967742  76.90000
```

Որպես երկրորդ փոփոխական կարող ենք փոխանցել նաև ֆակտորների ցուցակ, որը պարունակում է խմբավորման մեկից ավելի փոփոխականներ: *airquality* ուղղանկյունաձև փվյալներում ստեղծենք նոր փոփոխական, որը ցույց է տալիս թե օրը, որում կապարվել է չափումը գույգ օր է, թե՛ կենս: Այդ փոփոխականն էլ օգտագործենք որպես խմբավորման երկրորդ փոփոխական:

```
dat <- airquality
dat$Day_Type <- ifelse(dat$Day%%2==0, "Even", "Odd")

sapply(split(dat[,c("Ozone", "Solar.R", "Wind", "Temp")],
             list(dat$Month, dat$Day_Type)), colMeans, na.rm=TRUE)
```

```
##           5.Even    6.Even    7.Even    8.Even    9.Even    5.Odd
## Ozone    26.38462  23.250000  58.250000  51.928571  29.800000  20.84615
## Solar.R  187.42857 196.800000 234.933333 161.769231 158.333333 174.69231
## Wind     11.88000   9.633333   9.353333   9.426667   9.933333  11.38125
## Temp     65.73333  79.600000  84.466667  84.200000  77.066667  65.37500
##           6.Odd    7.Odd    8.Odd    9.Odd
## Ozone    34.4000  59.85714  69.33333  33.21429
## Solar.R  183.5333 199.18750 180.60000 176.53333
## Wind     10.9000   8.55625   8.20000   10.42667
## Temp     78.6000  83.37500  83.75000  76.73333
```

Եթե լինել ֆակտորների պիտակների համադրություն, որին համապատասխան արժեք չգտնվել, ապա *drop* արգումենտի *TRUE* արժեքի միջոցով հնարավոր կլինել դուրս գցել այդ համադրությունը՝ վերջնական արդյունքը ներկայացնելիս:

### 2.5.6 Խնդիրներ

1. Տրված ուղղանկյունաձև փվյալների համար պարզել յուրաքանչյուր սյան փեսակը:

#### Լուծում

```
airquality <- airquality[1:10,]
class(airquality)
```

```
## [1] "data.frame"
```

Պարզապես կիրառելով `class()` ֆունկցիան կարանանք ուղղանկյունաձև կառուցվածքի փեսակը: Փոխարենը պետք է կիրառել `class()` ֆունկցիան ուղղանկյունաձև փվյալների սյուների վրա: Կիրառենք `sapply()` ֆունկցիան: Պարզելու համար թե ինչպես կաշխարհի `sapply()` ֆունկցիան, վերածենք փրված ուղղանկյունաձև փվյալները ցուցակի՝

```
as.list(airquality)
```

```
## $Ozone
## [1] 41 36 12 18 NA 28 23 19 8 NA
##
## $Solar.R
## [1] 190 118 149 313 NA NA 299 99 19 194
##
## $Wind
## [1] 7.4 8.0 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6
##
## $Temp
## [1] 67 72 74 62 56 66 65 59 61 69
##
## $Month
## [1] 5 5 5 5 5 5 5 5 5 5
##
## $Day
## [1] 1 2 3 4 5 6 7 8 9 10
```

Այսպիսով փեսանում ենք, որ կիրառելով `sapply()` ֆունկցիան, այն կաշխարհի սյուն առ սյուն՝

```
sapply(airquality, class)
```

```
## Ozone Solar.R Wind Temp Month Day
## "integer" "integer" "numeric" "integer" "integer" "integer"
```

Կարող ենք օգտագործել նաև `apply()` ֆունկցիան՝

```
apply(airquality, 2, class)
```

```
## Ozone Solar.R Wind Temp Month Day
## "numeric" "numeric" "numeric" "numeric" "numeric" "numeric"
```

2. Տրված է ցուցակ, որը պարունակում է վեկտորներ: Մեր նպատակն է պարզել յուրաքանչյուր վեկտորի երկարությունը:

#### Լուծում

```
l <- list(x=c(11,0.98,-0.32,43), y=c("a", "cda", "12"), z="c")
```

```
l
```

```
## $x
## [1] 11.00 0.98 -0.32 43.00
##
```

```
## $y
## [1] "a" "cda" "12"
##
## $z
## [1] "c"
```

Եթե այս ցուցակի վրա կիրառենք `length()` ֆունկցիան, ապա կստանանք ցուցակի երկարությունը, ոչ թե ցուցակի յուրաքանչյուր ենթաօբյեկտի երկարությունը:

```
length(l)
```

```
## [1] 3
```

Փոխարենը՝

```
sapply(l, length)
```

```
## x y z
## 4 3 1
```

3. **airquality** ուղղանկյունաձև փվյալներում հաշվել յուրաքանչյուր ամսվա միջին ջերմաստիճանը, ըստ ամիսների, այնուհետև Օզոնի քանակը ըստ ամիսների, հիշելով, որ վերջին դեպքում փոփոխականը պարունակում է անհայտ արժեքներ:

**Լուծում**

```
tapply(airquality$Temp, airquality$Month, mean)
```

```
##      5
## 65.1
```

```
tapply(airquality$Ozone, airquality$Month, mean, na.rm = TRUE)
```

```
##      5
## 23.125
```

4. Սահմանել ֆունկցիա, որը վերադարձնում է փրված թվային փոփոխականի նմագագույն արժեքը, միջինը, կիսորդինը, միջին քառակուսային շեղումը և առավելագույն արժեքը՝ համասեռ վեկտորի տեսքով: Կիրառել այդ ֆունկցիան **airquality** ուղղանկյունաձև փվյալների Օզոն փոփոխականի վրա՝ ըստ ամիսների:

**Լուծում**

```
Stat <-function(x,...){
  y<-c(min(x,...), mean(x,...), median(x,...), sd(x,...), median(x,...))
  names(y) <- c("Min", "Mean", "Median", "Sd", "Median")
  y
}
```

```
tapply(airquality$Ozone, airquality$Month, Stat, na.rm=TRUE)
```

```
## $`5`
##      Min      Mean  Median      Sd  Median
## 8.0000 23.1250 21.0000 11.3696 21.0000
```

5. Էրապոսթենեսի մադի հետևյալ իրագործումը գրել առանց շրջապտույտի՝

```
rm(list=ls())
n<-100
Y<-seq_len(n)
Y<-Y[Y>1]
for(x in Y){
```



```
Y<-Y[Y%%x!=0|Y==x]
}
```

### Լուծում

Օգրագործելու ենք ֆունկցիա, որն աշխարհացնելիս գորբալ միջավայրում սահմանված համասեռ վեկտորից հեռացվում են պարզ թվեր:

```
n<-100
X<-seq_len(n)

Sieve <- function(x){
  if(x!=1){
    X<<-X[X%%x!=0|X==x]
  }
  else X<<-X[X>1]
}

l<-lapply(1:n,Sieve)
X
```

```
## [1] 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
## [24] 89 97
```

```
identical(X,Y)
```

```
## [1] TRUE
```

6. Վերասահմանել ստորև բերված ֆունկցիան օգրվելով < <- վերագրման գործողությունից, այնպես որ ֆունկցիայի սահմանման մեջ այլևս *for* կառուցվածք չլինի:

```
rm(list=ls())

Add_Col <- function(dat, NR){
  x<-NR["Variable"]
  for(i in x){
    dat[,paste0(i,"FL")]<-ifelse(dat[,i]<NR[NR$Variable==i,"Low"],"Below",
                                ifelse(dat[,i]<=NR[NR$Variable==i,"High"],
                                        "Normal", "High"))
  }
  dat
}
```

### Լուծում

Նման դեպքերում ամենապարզ մարպավարությունը *for* կառուցվածքը մեկ արգումենտանոց ֆունկցիայով փոխարինելն է, այնպես որ այդ արգումենտը կապարի *for* կառուցվածքի կրկնման փոփոխականի դերը, և կիրառել այդ ֆունկցիան կրկնման փոփոխականի փոփոխման փիրույթը ցույց տվող վեկտորի վրա՝ *apply* ընդդրանիքի ֆունկցիաներից մեկի միջոցով:

```
dat<-airquality
NR<-data.frame(Variable=c("Wind", "Temp"), Low=c(7,70), High=c(10,80))

x<-as.character(NR["Variable"])

Add_Col2 <- function(i){
  dat[,paste0(i,"FL")]<<-ifelse(dat[,i]<NR[NR$Variable==i,"Low"],"Below",
```

```

        ifelse(dat[,i]<=NR[NR$Variable==i,"High"],
              "Normal", "High"))
}

s<-sapply(x,Add_Col2)

```

Ինչպես տեսնում ենք՝ ուղղանկյուն փյուռիկները որպես արգումենտ չենք փոխանցում մեր սահմանած ֆունկցիային, քանի որ վերջինս հնարավորություն ունի ոչ միայն ստանալու գլոբալ միջավայրից արժեքներ, այլև փոփոխություններ կատարել գլոբալ միջավայրում:

Սրորև բերված վերագրման նպատակը գործողությունների կատարման ընթացքը թաքցնելն է: Մենք չունենք այդ փոփոխականի կարիքը:

```
s<-sapply(x, Add_Col2)
```

Իրականում նոր ուղղանկյուն փյուռիկ չի ստեղծվել, այլ գոյություն ունեցողի վրա կատարվել են անհրաժեշտ փոփոխությունները:

```
head(dat)
```

```
##   Ozone Solar.R Wind Temp Month Day WindFL TempFL
## 1    41    190  7.4   67     5  1 Normal  Below
## 2    36    118  8.0   72     5  2 Normal  Normal
## 3    12    149 12.6   74     5  3   High Normal
## 4    18    313 11.5   62     5  4   High  Below
## 5    NA     NA 14.3   56     5  5   High  Below
## 6    28     NA 14.9   66     5  6   High  Below
```

```
table(dat$WindFL)
```

```
##
## Below   High Normal
##    33    72    48
```

```
table(dat$TempFL)
```

```
##
## Below   High Normal
##    32    68    53
```

```
table(dat$WindFL, dat$TempFL)
```

```
##
##           Below High Normal
## Below      0    22    11
## High       22    20    30
## Normal     10    26    12
```

- airquality** ուղղանկյունաձև փյուռիկներում Ջերմաստիճանը համարվում է նորմալ, եթե այն գրնվում է [68, 86] միջակայքում: Եթե Ջերմաստիճանը ցածր է այս միջակայքից, ապա Ջերմաստիճանը համարվում է ցուրտ, եթե բարձր է, ապա՝ տաք: Նաշվել Ջերմաստիճանի ամեն դասակարգման համար Օզոնի լայնքի միջնակետը (լայնքի միջնակետը նվազագույն և առավելագույն արժեքների միջին թվաբանականն է): Օզոնագործել անանուն ֆունկցիա: Լուծել նույն խնդիրը, բայց այս անգամ դասակարգումը վերցնել ըստ ամսիների, իսկ ամիսների ներսում՝ ըստ Ջերմաստիճանի դասակարգման:

### Լուծում

```

dat<-airquality
dat$Temp.C<-ifelse(dat$Temp<68,"Cold",ifelse(dat$Temp<=86,"Normal","Hot"))
tapply(dat$Ozone,dat$Temp.C, function(x,...) mean(range(x,...)),na.rm=T)

```

```
## Cold Hot Normal
## 21.0 80.5 87.5
```

```
tapply(dat$Ozone, list(dat$Month, dat$Temp.C),
        function(x, ...) mean(range(x, ...)), na.rm=T, drop=T)
```

```
## Cold Hot Normal
## 5 21.0 NA 58.0
## 6 10.5 36.0 19.0
## 7 NA 49.0 68.0
## 8 NA 61.5 84.5
## 9 9.5 48.5 23.5
```

8. Գրել անանուն ֆունկցիա, որն ունի մեկ արգումենտ: Որպես արգումենտ այդ ֆունկցիան ստանում է համասեռ վեկտոր: Եթե այդ վեկտորը թվային չէ, ապա ֆունկցիան ոչինչ չի վերադարձնում: Եթե այդ վեկտորը թվային է, ապա ֆունկցիան վերադարձնում է այդ վեկտորի երկրորդ ամենափոքր արժեքը: Կիրառել այդ ֆունկցիան *airquality* ուղղանկյուն տվյալների սյուների վրա:

#### Լուծում

```
apply(airquality, 2, function(x) if(is.numeric(x)) return(sort(x)[2]))
```

```
## Ozone Solar.R Wind Temp Month Day
## 4.0 8.0 2.3 57.0 5.0 1.0
```

### 2.5.7 Ամփոփում

- Loop Functions

#### Functions

- *lapply()*, *sapply()*
- *apply()*
- *rowSums()*, *rowMeans()*, *colSums()*, *colMeans()*
- *vapply()*, *mapply()*, *tapply()*

## 2.6 Տվյալների հեք աշխատանք

### 2.6.1 Ամսաթիվ և ժամանակային տվյալներ

**R**-ում առկա է ամսաթվերի և ժամանակային ներկայացման հարույկ ձև: Ամսաթվերը ներկայացվում են *Date* տեսակի միջոցով, իսկ ժամանակային տվյալները՝ *POSIXlt* և *POSIXct* տեսակների միջոցով: **R**-ի ներսում ամսաթվերը պահվում են որպես 1970 – 01 – 01 ամսաթվից մինչ փրված ամսաթիվն ընկած օրերի քանակը: Իսկ վայրկյանները ներկայացվում են որպես կան այդ նույն սկզբնական ամսաթվից վայրկյանների քանակ՝ *POSIXct* կամ որպես վայրկյանների, րոպեների և ժամերի ցուցակ՝ *POSIXlt*: Սկսենք հերևյալ օրինակից՝

```
d1 <- Sys.Date()
class(d1)
```

```
## [1] "Date"
```

Օգտագործելով *unclass()* ֆունկցիան կարող ենք տեսնել թե ինչպես է այս արժեքը պահվում **R**-ի ներսում՝

```
unclass(d1)
```

```
## [1] 17865
```

Բայց պատվով *d1* փոփոխականի արժեքը կրենանք, որ այն ունի ներկայացման հարուկ ձև՝

```
d1
```

```
## [1] "2018-11-30"
```

Ամսաթիվ տեսակի արժեք սրանալու ամենապարզ ձևը տեքստային տեսքով այդ ամսաթիվը գրելն է և այն *as.Date()* ֆունկցիային փոխանցելը՝

```
D1 <- "1991-09-21"
class(D1)
```

```
## [1] "character"
```

```
D1_ <- as.Date(D1)
class(D1_)
```

```
## [1] "Date"
```

```
D1_
```

```
## [1] "1991-09-21"
```

```
unclass(D1_)
```

```
## [1] 7933
```

Ենթադրենք ունենք ամսաթիվ, որը նախորդում է սկզբնակետ հանդիսացող ամսաթիվը: Այդ դեպքում՝

```
d2 <- as.Date("1969-01-01")
unclass(d2)
```

```
## [1] -365
```

Մրացվում է բացասական թիվ, որի նշանը և ցույց է տալիս, որ տրված ամսաթիվը սկզբնական հանդիսացող ամսաթիվին նախորդում է: Դիտարկենք ժամանակային փվյալները:

```
t1 <- Sys.time()
class(t1)
```

```
## [1] "POSIXct" "POSIXt"
```

```
t1
```

```
## [1] "2018-11-30 08:05:51 +04"
```

```
unclass(t1)
```

```
## [1] 1543550751
```

*Sys.time()* ֆունկցիան վերադարձնում է *POSIXct* տեսակի արժեք, որը կարող ենք փոխարկել *POSIXlt*-ի՝

```
t2 <- as.POSIXlt(Sys.time())
class(t2)
```

```
## [1] "POSIXlt" "POSIXt"
```

```
t2
```

```
## [1] "2018-11-30 08:05:51 +04"
```

```
unclass(t2)
```

```
## $sec
```

```
## [1] 51.15843
```

```
##
## $min
## [1] 5
##
## $hour
## [1] 8
##
## $mday
## [1] 30
##
## $mon
## [1] 10
##
## $year
## [1] 118
##
## $wday
## [1] 5
##
## $yday
## [1] 333
##
## $isdst
## [1] 0
##
## $zone
## [1] "+04"
##
## $gmtoff
## [1] 14400
##
## attr(,"tzone")
## [1] "" "+04" "+04"
```

Ինչպես նշվեց՝ սա ցուցակ է, որը պարունակում է ամսաթվի և ժամանակի վերաբերյալ բոլոր տվյալները: Դրա կառուցվածքն ավելի լավ ուսումնասիրելու համար կիրառենք `str()` ֆունկցիան՝

```
str(unclass(t2)) #gives the structure of the object
```

```
## List of 11
## $ sec : num 51.2
## $ min : int 5
## $ hour : int 8
## $ mday : int 30
## $ mon : int 10
## $ year : int 118
## $ wday : int 5
## $ yday : int 333
## $ isdst : int 0
## $ zone : chr "+04"
## $ gmtoff: int 14400
## - attr(*, "tzone")= chr [1:3] "" "+04" "+04"
```

Քանի որ վերադարձվող կառուցվածքը ցուցակ է, ապա դրանից հեշտությամբ կարող ենք ենթակառուցվածք ստանալ, օգտվելով ցուցակի ենթաօբյեկտների անուններից՝

```
t2$min
```

```
## [1] 5
```

Գոյություն ունեն ֆունկցիաներ, որոնք որպես արգումենտ ընդունում են ամսաթիվ կամ ժամանակային փոփոխության արժեքներ և որոշակի փոփոխություն են հանում դրանցից: Պարզելու համար թե շաբաթվա որ օրն է փրկված ամսաթիվը՝ կիրառենք `weekdays()` ֆունկցիան՝

```
weekdays(d1)
```

```
## [1] "Friday"
```

`months()` ֆունկցիան վերադարձնում է փրկված ամսաթվի անիսը՝

```
months(d1)
```

```
## [1] "November"
```

`quarters()` ֆունկցիան վերադարձնում է այն քառամսյակը որում գտնվում է փրկված ամսաթիվը՝

```
quarters(t2)
```

```
## [1] "Q4"
```

Նաճախ փոփոխություն կարողանա ամսաթիվը բացահայտորեն փրկված չէ: Այդ դեպքում կարող է օգտագործվել `strptime()` ֆունկցիան, որը փոփոխության փոփոխություն կվերափոխի `POSIXt` փոփոխության անգամ եթե այդ փոփոխության փոփոխություն փոփոխության փոփոխություն փոփոխություն փոփոխություն:

```
t3 <- "October 17, 1986 08:24"
t4 <- strptime(t3, "%B %d, %Y %H:%M")
class(t4)
```

```
## [1] "POSIXlt" "POSIXt"
```

```
t4
```

```
## [1] "1986-10-17 08:24:00 +04"
```

Քանի որ `R`-ը ամսաթիվերը և ժամանակային փոփոխությունը պահում է որպես թվեր, ապա դրանց հեղափոխումը հնարավոր է թվարանական և փրամարանական գործողություններ կատարել՝

```
Sys.time() > t1
```

```
## [1] TRUE
```

```
Sys.time() - t1
```

```
## Time difference of 0.0785861 secs
```

Ինչպես տեսնում ենք՝ ժամանակային փոփոխությունը փրկվում է վայրկյաններով: Որպեսզի կարողանանք այդ փոփոխությունը սրանալ մեզ համար անհրաժեշտ միավորով՝ կարող ենք օգտագործել `difftime()` ֆունկցիան՝

```
difftime(Sys.time(), t1, units = 'days')
```

```
## Time difference of 9.848619e-07 days
```

Ամսաթիվ հանդիսացող փոփոխականը հնարավոր չէ համեմատել ժամանակային փոփոխականի հեղափոխումը: Այդ պարագայում պետք է փոփոխություն փոփոխության փոփոխություն՝ մինչև համեմատումը:

```
x1 <- as.Date("1991-09-21")
x2 <- Sys.time()
x2 - as.POSIXct(x1)
```

```
## Time difference of 9932.171 days
```

Իսկ `difftime()` ֆունկցիային որպես արգումենտ փոխանցելիս փոխարկումը կատարվում է ֆունկցիայի ներսում՝

```
x1 <- as.Date("1991-09-21")
x2 <- Sys.time()
difftime(x2,x1,units = "weeks")
```

```
## Time difference of 1418.882 weeks
```

Ամսաթվերը և ժամանակային տեսակները հաշվի են առնում նահանջ տարիները և ժամային գոտիները: Ենթադրենք դուք Երևանից մեկնել եք Նյու Յորք և գրացել եք ժամանակը մինչ թռիչքը և վայրէջքից հետո, ապա կարող եք հաշվել թե ինչքան է տևել թռիչքը:

```
bef <- as.POSIXct("2018-11-17 04:00:00", tz="Asia/Yerevan")
aft <- as.POSIXct("2018-11-17 07:30:00", tz="EST")
aft - bef
```

```
## Time difference of 12.5 hours
```

### 2.6.2 Տեքստային տվյալների հեղի աշխատանք

Ինչպես նշել ենք, տեքստային տվյալներ պարունակող փոփոխականի երկարությունը 1 է, բայց կարող ենք հաշվել տառատեսակների քանակը՝

```
x<-"This is a very long sentence!"
length(x)
```

```
## [1] 1
```

```
nchar(x)
```

```
## [1] 29
```

Տեքստային փոփոխականների հեղի աշխատելու հաջորդ ֆունկցիան, որ օգտագործել ենք `paste()` ֆունկցիան է: Այն միավորում էր տրված տեքստային փոփոխականները մեկ տեքստային փոփոխականի մեջ, կամ տեքստային համասեռ վեկտորը սեղմում էր և դարձնում մեկ համասեռ, տեքստային փոփոխական:

```
paste("Name", "Surname", sep="-")
```

```
## [1] "Name-Surname"
```

```
x<-c("Name", "Surname")
paste(x, collapse = "-")
```

```
## [1] "Name-Surname"
```

Թեև տեքստային փոփոխականն իրականում 1 երկարությամբ համասեռ վեկտոր է, բայց, այնուամենայնիվ, կարող ենք վերլուծել դրանում առկա տեղեկությունները:

Նախևառաջ կարող ենք տեքստը վերածել ամբողջությամբ մեծատառերով կամ փոքրատառերով տեքստի՝

```
toupper(x)
```

```
## [1] "NAME" "SURNAME"
```

```
tolower(x)
```

```
## [1] "name" "surname"
```

Նախորդ ֆունկցիան է `substr()` ֆունկցիան, որը թույլ է տալիս տրված տեքստային փոփոխականից մասեր առանձնացնել: Այն ունի երեք արգումենտ՝ տեքստային տվյալը, որն ուզում ենք մասնատել, տառատեսակի դիրքը, որից սկսելու ենք և այն տառատեսակի դիրքը, որով ավարտելու ենք:

```
substr(x,start=1,stop=10)
```

```
## [1] "Name"      "Surname"
```

Այս ֆունկցիայի միջոցով հնարավոր է նաև փոխել տեքստային փոփոխականի որոշ տառաբաղաձայնների արժեքները՝

```
x<-"This is a very long sentence!"
substr(x,1,1)<-"t"
x
```

```
## [1] "this is a very long sentence!"
```

Տեքստային փոփոխականը մասնատելու և մասերից բաղկացած համասեռ վեկտոր սպանալու համար օգտագործվում է *strsplit()* ֆունկցիան, որը կապարում է *paste()* ֆունկցիայի *collapse* արգումենտի հակառակ գործառույթը՝

```
x
```

```
## [1] "this is a very long sentence!"
```

```
l<-strsplit(x," ")
l
```

```
## [[1]]
## [1] "this"      "is"        "a"         "very"      "long"      "sentence!"
```

```
class(l)
```

```
## [1] "list"
```

Կան նաև ֆունկցիաներ որոնք թույլ են տալիս պարզելու թե նշված տառաբաղաձայնը պարկանում է տեքստային փոփոխականին, թե՛ ոչ:

Առաջին երկու ֆունկցիաները՝ *startsWith()*, *endsWith()* ստուգում են, համապատասխանաբար, արված տառաբաղաձայնով բառի սկսվելը կամ ավարտվելը: Երրորդ ֆունկցիան՝ *grepl()* ստուգում է առհասարակ այդ տառաբաղաձայնի առկայությունը բառի մեջ, ընդ որում կարող ենք ստուգել ուշադրություն չդարձնելով մեծատառ կամ փոքրատառ լինելուն՝ *ignore.case = T*,

```
startsWith(x,"T")
```

```
## [1] FALSE
```

```
endsWith(x,".")
```

```
## [1] FALSE
```

```
grepl("T",x)
```

```
## [1] FALSE
```

```
grepl("T",x,ignore.case = T)
```

```
## [1] TRUE
```

*gregexpr()* ֆունկցիան ստուգում է, թե որ դիրքում է գտնվում արված տառաբաղաձայնը (վերադարձնում է -1, եթե տառաբաղաձայնը չի պարունակվում բառում): Եթե մի քանի համընկնում կա վերադարձվում են բոլոր դիրքերը, իսկ *regexpr()* ֆունկցիան միայն վերադարձնում է առաջին համընկած տառաբաղաձայնի դիրքը:

```
l<-gregexpr("i",x)
l
```

```
## [[1]]
```

```
## [1] 3 6
```



```
## attr(,"match.length")
## [1] 1 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE

l0<-as.numeric(l[[1]]) #coercing drops all attributes
l0
```

```
## [1] 3 6
```

Ընդ որում `regexpr()` ֆունկցիան վերադարձնում է համասեռ վեկտոր, իսկ `gregexpr()` ֆունկցիան վերադարձնում է ցուցակ:

Նիշենք, որ արեսակի հարկադիր փոփոխման արդյունքում օբյեկտի բոլոր հարկանիշները կորում են, ինչը հնարավորություն է տալիս պահել միայն մեզ համար անհրաժեշտ դիրքերի համարները՝

```
u<-regexpr("i",x)
u
```

```
## [1] 3
## attr(,"match.length")
## [1] 1
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

```
as.numeric(u)
```

```
## [1] 3
```

Որպես օրինակ կարող ենք դիտարկել տեքստային փայլի՝ փրկած փառատեսակների միջև ընկած հարվածը կտրելու խնդիրը՝

```
x <- "email:name.surname@mail.com"
i1 <- as.numeric(regexpr(":",x))
i2 <- as.numeric(regexpr("@",x))
substr(x,i1+1,i2-1)
```

```
## [1] "name.surname"
substr(x,i1+1,nchar(x))
```

```
## [1] "name.surname@mail.com"
```

### 2.6.3 Տվյալների կարգացումը համակարգչից և գրումը համակարգչում

Որպեսզի հնարավոր լինի կատարել փայլների վերլուծություն պետք է վերբեռնել փայլները **R**-ի մեջ՝ դրանք կարդալու միջոցով: Բայց, նախևառաջ, պետք է կապ հաստատել **R**-ի և փայլները պարունակող համակարգչի պանակի միջև: **R**-ը սկզբնապես ունի պանակ, որի հեղափոխ է հաստատված: Այն կոչվում է հասցեապանակ՝ `directory`: Դա կարելի է տեսնել `getwd()` ֆունկցիայի միջոցով՝

```
getwd()
```

Ցանկացած նոր ծրագիր գրելիս պետք է ստեղծել առանձին հասցեապանակ, այնպեղ տեղադրել բոլոր փայլները, որոնք օգտագործվելու են վերլուծության համար, ինչպես նաև այնպեղ գրել հաշվարկների բոլոր արդյունքները՝ վիճակագրական վերլուծություններն ավարտելուց հետո: `setwd()` ֆունկցիայի միջոցով

հնարավոր է փոխել հասցեապանակը: Ընդամենը պետք է նոր պանակ ստեղծել և դրա հասցեն որպես արգումենտ փոխանցել `setwd()` ֆունկցիային:

```
R.version.string #The result of execution of this code should be copied in the code
```

```
## [1] "R version 3.5.0 (2018-04-23)"
```

```
##"R version 3.5.0 (2018-04-23)"
```

```
rm(list=ls())
```

```
path <- paste(getwd(),"Data",sep="//")
setwd(path)
```

Պետք է հիշել, որ **R**-ում պարամետրերի ենթապանակները սրանալու համար օգտագործվում է կրկնակի թեք գիծ (թեքությունը՝ ձախ) նշանը, որը փարբերվում է Windows օպերացիոն համակարգի բաժանիչից, հեքսաբար, հաճախ, պարզեցնելով պանակի հասցեն Windows-ից և այն բերելով **R** սխալ է տրեղի ունենում: Որպեսզի առհասարակ չհիշել փարբեր օպերացիոն համակարգերում բաժանիչների փարբերության մասին կարելի է օգտագործել `file.path()` ֆունկցիան՝ պանակի հասցեն սրանալու համար: Փոխանցելով այդ ֆունկցիային միայն իրար մեջ ներդրված պանակների անունները այն կկառուցի անհրաժեշտ պանակի հասցեն:

```
path <- file.path(getwd(),"Data")
setwd(path)
```

Բացի այդ, մինչ որևէ փոխյալ ներբեռնելը պետք է ստուգել, որ նշված փոխյալներն առկա են հասցեապանակում: Հասցեապանակի ամբողջ պարունակությունը կարելի է ստուգել `dir()` ֆունկցիայի միջոցով: Այդ ֆունկցիան, ինչպես և `getwd()` ֆունկցիան, ոչ մի արգումենտ չունի:

Հանդգնելով որ անհրաժեշտ փոխյալները գրվում են հասցեապանակում՝ կարող ենք դրանք կարդալ: Տվյալները սովորաբար պահվում են համակարգում ստորակետներով անջատված արժեքների՝ `csv` (comma separated values) ձևաչափով: Դա հնարավոր է կարդալ `read.csv()` ֆունկցիային որպես առաջին արգումենտ փոխյալ անունը, որով փոխյալները պահված են հասցեապանակում և այն կհայտնվի **R**-ում ուղղանկյունաձև կառուցվածքի տեսքով:

```
dat <- read.csv("Data//PEP_2017_PEPANNRES_with_ann.csv",stringsAsFactors = FALSE)
class(dat)
```

```
## [1] "data.frame"
```

Պետք է ուշադիր լինել, որ անունը գրելիս վերջում անպայման նշված լինի `csv` տեսակը:

`read.csv()` ֆունկցիան բացի փոխյալները պարունակող ֆայլի անունից ունի նաև այլ արգումենտներ: `stringsAsFactors` արգումենտը ցույց է տալիս թե արդյոք տեքստային փոխյալներ պարունակող սյուները պետք է ֆակտոր դառնան թե ոչ: Այս արգումենտի լռելյալ արժեքը `TRUE` է, որը նշանակում է, որ փոխյալները կարդալիս տեքստային փոխյալներ պարունակող բոլոր սյուները դառնում են ֆակտոր: Սա վերլուծություններ կատարելիս հաճախ բերում է անսպասելի արդյունքների, այդ պատճառով անհրաժեշտ է փոխյալները կարդալիս այս արգումենտի արժեքը փոխել `FALSE`-ի, ինչը և կատարված է վերևում:

Դիտելով ստեղծված ուղղանկյունաձև փոխյալների կառուցվածքի առաջին մի քանի տողերն ու սյուները նկատում ենք, որ դրանում պահված են ԱՄՆ նահանգների բնակչությունները ըստ փարիների: Բայց խնդիր կա՝ `GEO.display.label` սյունից տեսնում ենք, որ նահանգների հեղափոխ պահված է ԱՄՆ բնակչությունն ամբողջությամբ, հեքսաբար եթե, օրինակ, ուզենանք հաշվել որևէ թվականին ԱՄՆ նահանգների միջին բնակչությունը, ապա սխալ արդյունք կստանանք: Դրա համար հեռացնենք այդ տողը, ինչպես նաև `GEO.id2` սյունը՝

```
dat[1:6, 1:4]
```

```
##      GEO.id GEO.id2 GEO.display.label rescen42010
## 1  0100000US      NA      United States  308745538
## 2  0400000US01      1           Alabama    4779736
## 3  0400000US02      2           Alaska    710231
```

```
## 4 0400000US04      4      Arizona      6392017
## 5 0400000US05      5      Arkansas      2915918
## 6 0400000US06      6      California    37253956
```

```
dat0 <- dat[dat$GEO.display.label!="United States",!names(dat)%in%c("GEO.id2")]
dat0[1:6, 1:4]
```

```
##      GEO.id GEO.display.label rescen42010 resbase42010
## 2 0400000US01      Alabama      4779736      4780135
## 3 0400000US02      Alaska       710231       710249
## 4 0400000US04      Arizona      6392017      6392309
## 5 0400000US05      Arkansas     2915918      2916031
## 6 0400000US06      California   37253956     37254518
## 7 0400000US08      Colorado     5029196     5029325
```

Սրացված տվյալները կարող ենք պահել՝ հեղազա վերլուծություններում օգտագործելու համար: Սա կարող ենք անել `write.csv()` ֆունկցիայի միջոցով:

#### 2.6.4 Խնդիրներ

#### 2.6.5 Անփոփում

- Directory

##### Functions

- `getwd()`, `setwd()`
- `dir()`
- `file.path()`
- `read.csv()`, `write.csv()`
- `unclass()`
- `as.Date()`
- `Sys.Date()`, `Sys.time()`
- `as.POSIXlt()`, `as.POSIXct()`
- `str()`
- `weekdays()`
- `months()`
- `quarters()`
- `strptime()`
- `difftime()`
- `toupper()`, `tolower()`
- `substr()`
- `strsplit()`
- `startsWith()`, `endsWith()`
- `grepl()`
- `gregexpr()`, `regexpr()`

**2.7 Կիրառական վիճակագրություն**

**2.7.1** Սկզբնական վիճակագրական գաղափարներ

**2.7.2** Նիմնական գծապարկերների տեսակները

**2.7.3** Խնդիրներ

**2.7.4** Ամփոփում

**3 Տվյալագիտական եզրույթներ**

---