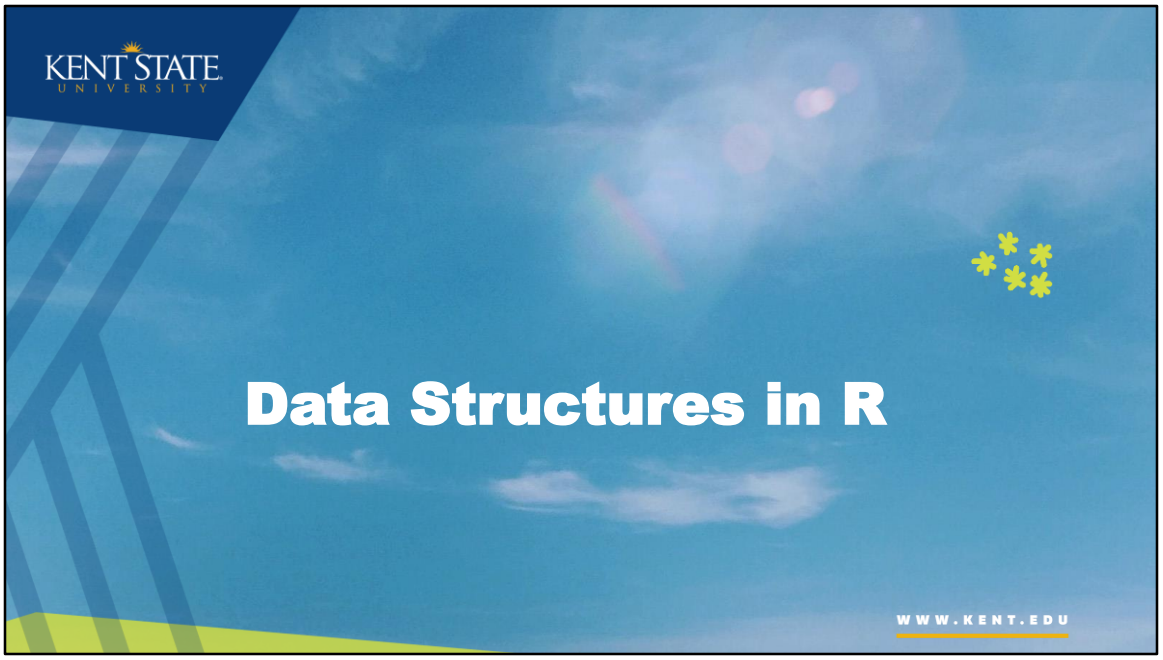


Data structure can be defined as the specific form of organizing and storing the data. R programming supports five basic types of data structure . This module introduces these data structures and provides examples.

KENT STATE UNIVERSITY

Value Assignment

- At the R prompt we type expressions. The <- symbol is the assignment operator.

```
x <- 5
print(x) #printing the value of x
```

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

```
x
```

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

```
msg <- "hello world"
```

- The # character indicates a comment. Anything to the right of the # (including the # itself) is ignored.

WWW.KENT.EDU

Before we start considering data structures in R, let's see how we can assign values to variables. A variable is a symbolic name associated with a value where the associated value may be changed. For example, let's say we want to define a variable x which has a value of 5. The assignment operator in R consists of a "Lesser than" sign and a dash sign which will form a right to left arrow. The assignment of 5 to variable x is shown in this slide. Note that in R, we are not required to declare a variable before assigning a value to it. We can also use the same syntax for assigning string values to variables as shown. The print() function can be used to print the value of a variable. Finally, note that hashtag (#) is a character that indicates a comment. In other words, whatever comes after a hashtag is not compiled. Comments are important to improve readability of the code.

KENT STATE UNIVERSITY

Evaluation

- When a complete expression is entered at the prompt, it is evaluated and the result of the evaluated expression is returned. The result may be auto-printed.

```
x <- 10  # nothing printed
x        # auto-printing occurs
```

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

```
print(x) # explicit printing
```

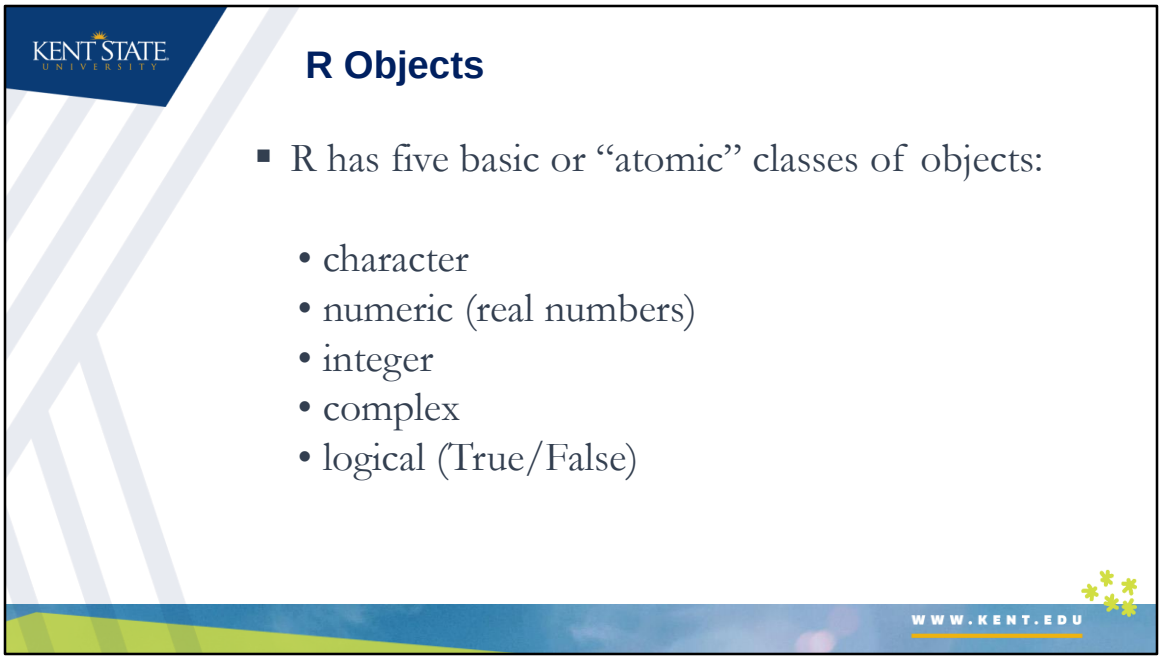
```
## [1] 10
```

```
y=1:5
y
```

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

WWW.KENT.EDU

When a complete expression is entered at the prompt, it is evaluated and the result of the evaluation expression is returned. This result may be auto-printed as well. So consider a simple example; in this example, we are assigning value ten to variable X. if we print the name of X, the value that is assigned to X is auto-printed, in this case it's ten. We can also use explicit printing by using the print function.



R has five basic atomic classes of objects. This includes character values, numeric values, or real numbers, integer numbers, complex numbers, as well as the logical values which can take true or false values. We will provide some examples in the following side.

KENT STATE UNIVERSITY

Examples

```
x1<-"Hello"
class(x1)
## [1] "character"

x2<-10
class(x2)
## [1] "numeric"

x3<-10L
class(x3)
## [1] "integer"

x4<-10+2i
class(x4)
## [1] "complex"

x5<-TRUE
class(x5)
## [1] "logical"
```

WWW.KENT.EDU

Her we provide 5 examples of basic or atomic data objects in R. In the first example, we assign string hello to variable x1. We can use the function class to determine the class of a variable. In this case, class x1 returns character. If we assign a number to a variable, by default, the variable has a class of numeric, so in this case we can see if we assign ten to x2, the class of x2 is numeric. If we want that number to be explicitly coded as integer, we need to provide L at the end of the number. So, in this case if we assign 10L x3, the class of x3 would be integer. Also, as you can see, x4 would be a complex number which contains 10 plus 2i, which two would be the imaginary part of the complex number and 10 would be the real part of the complex number. If we assign true or false to a given variable, the class of the variable would be logical.

KENT STATE UNIVERSITY

R Data Structures

- R also has many data structures. These include
  - Atomic Vectors
  - List
  - Matrix
  - Data frame
  - Factors
  - Tables

WWW.KENT.EDU

R also has many data structures, these include atomic vectors, list, matrix, data frame, factors and tables. We will discuss these data structures in the following slides.

Atomic Vectors

- An atomic vector can be a vector of characters, logical, integers, numeric or complex values.
- The c() function can be used to create vectors of objects by concatenating things together
- Unlike lists, a vector can only contain objects of the same class.

```
x <- c(0.5, 0.6) ## numeric
x <- c(TRUE, FALSE) ## logical
x <- c(T, F) ## logical
x <- c("a", "b", "c") ## character
x <- 9:29 ## integer
x <- c(1+0i, 2+4i) ## complex
```


The first data structure is atomic vector; an atomic vector can be a vector of characters, logical, integers, numeric or complex values. What is important here is that the atomic vector can only contain objects of the same class. So, you can have a vector of characters, or a vector of logical values or a vector of integers, but you cannot have mixed values in a vector. The function C can be used to create vectors of objects by combining different elements together, So, here are some examples. The first example is a vector of numerics which contains two specific values of 0.5 and 0.6. We use the function C to combine to values and store them in the variable X. So, in the case variable X would be a vector of numeric values.

Mixing Objects

- There are occasions when different classes of R objects get mixed together. Sometimes this happens by accident but it can also happen on purpose.
- When different objects are mixed in a vector, implicit coercion occurs so that every element in the vector is of the same class.

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

## [1] "numeric"

x <- c(1, 2,"Alan")
x

## [1] "1"    "2"    "Alan"

class(x)

## [1] "character"
```

WWW.KENT.EDU

So, you might ask what would happen if we start mixing objects in an atomic vector. Well in some cases you might get an error, but there are other cases where R may decide to automatically change the class of the vector. Consider this example, X is a vector containing values one and two, the class of X would be obviously numeric. Now imagine the same example where we add another element, which is a string as Alan. In this case, instead of showing an error, R automatically converts one and two to string values and the class of X would be character. This is referred to as coercion, which can happen if R is capable of converting the classes.

KENT STATE UNIVERSITY

Explicit Coercion

- Objects can be explicitly coerced from one class to another using the as.* functions, if available.

```
 x <- 0:6
class(x)
## [1] "integer"

as.numeric(x)
## [1] 0 1 2 3 4 5 6

as.logical(x)
## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE

as.character(x)
## [1] "0" "1" "2" "3" "4" "5" "6"

x <- c("a", "b", "c")
as.numeric(x)
## Warning: NAs introduced by coercion
## [1] NA NA NA
```

WWW.KENT.EDU

you can also explicitly convert one data type to another. For example, as the numeric function converts the input variable of the function to numeric values. Similarly, as the logical converts the input of the function into the logical value. Just note that the conversion of all data types might not be possible. Note that the conversion between all data types might not be possible. For example, if you have a character type of A, B, and C, you may not be able to convert them into numeric. In that case you would get a warning and NA values would be produced.

KENT STATE UNIVERSITY

Matrices

- Matrices are two dimensional vectors . The shape of a matrix is defended by two integers (number of rows, number of columns)

```
m <- matrix(nrow = 2, ncol = 3)
m
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA

dim(m)
## [1] 2 3

m <- matrix(1:6, nrow = 2, ncol = 3)
m
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

WWW.KENT.EDU

Matrices are two dimensional vectors. The shape of a matrix is defined by two integer values, the number of rows and the number of columns. So, we can define matrixes just by defining the number of rows and the number of columns. So, in this example, m is a matrix where the number of rows is two and the number of columns is three. Because the values of the m have not been defined, NA values are assigned by default. We can use the dim function to see the dimensions of the m matrix. In this case, the dim m returned two and three where two is the number of rows and three is the number of columns of the matrix. We can also assign values to the matrix when performing matrixes as well. So, in the second example, m is the matrix which has the values from one to six and consists of two rows and three columns. So, as you can see the first column of m would be one and two, the second column would be three and four, and the third column would be five and six.

KENT STATE UNIVERSITY

Lists

- Lists are a special type of vector that can contain elements of different classes. Lists can be explicitly created using the list() function.

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

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

WWW.KENT.EDU

Lists are considered as special types of vectors, and can contain elements of different classes, so you can have a list that contains an integer, a character and a complex number.

KENT STATE UNIVERSITY

Factors

- Factors are used to represent categorical data and can be unordered or ordered. Factor objects can be created with the factor() function.

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

```
## [1] yes yes no  yes no
## Levels: no yes
```

```
table(x)
```

```
## x
##  no yes
##   2   3
```

WWW.KENT.EDU

Factors are used to represent categorical data and can be unordered or ordered. Factors are important element of many statistical modeling methods.

Using factors with labels is *better* than using integers because factors are self-describing. For example, having a variable that has values “Male” and “Female” is better than a variable that has values 1 and 2. Factor objects can be created with the factor() function. You can use the table() function to see the frequency of each of the category levels.

Often factors will be automatically created for you when you read a dataset by using a function like read.table(). Those functions often default to creating factors when they encounter data that look like characters or strings.

Data Frames

- Data frames are used to store tabular data in R. They are an important type of object in R and are used in a variety of statistical modeling applications.

```
x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
x
##   foo   bar
## 1   1  TRUE
## 2   2  TRUE
## 3   3 FALSE
## 4   4 FALSE

x$foo
## [1] 1 2 3 4

colnames(x)
## [1] "foo" "bar"
```

WWW.KENT.EDU

Data frames are used to store tabular data in R. They are the most important type of object in R and are used in a variety of statistical modeling applications. Unlike matrixes, data frames can store different classes of objects in each column. For data frames, we can define names for columns, and refer to those names or variable names when doing statistical analyses. Let's consider a simple example in here. In here we define the data frame, which is two variables, the first variable is called foo and the second variable is called bar, the first variable contains integer values from one to four, and the second column or the second variable contains logical values of true, true, false, and false, we can use the dollar sign to get access to specific variables. So, in the case, x$ foo, will print out the values of the foo variable, which is one, two, three, and four, we can also use the column name functions, to check the name of the columns of a data frame.