Basics

cdurso

Please run each code chunk below, noting the output and the effects on the "Environment" window. Where asked, please supply your own code.

Hello World

There has to be a "Hello World".

Note that both the "print" statement and the variable name alone cause the contents of the variable to be shown as output.

```r
hello<-"Hello, World!"
hello
```

```
## [1] "Hello, World!"
```

```r
print(hello)
```

```
## [1] "Hello, World!"
```

Variables

A wide variety of data may be stored in variables. For example, one of the strengths of R is that a data set may be stored in a single variable. Multiple data sets may be active at the same time. Use of variables in ways familiar from intro programming is also available

Simple Variables

Variable names may use letters, numbers, ".", and "_". Names should not begin with a number and are case-sensitive.

By custom "<-" is used for assignment, though "=" also works.

The variables and some information regarding their values can be seen in the "Environment" window.

```r
first<-1
First<-2
first
```

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

```r
First
```

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

In the code block below, please assign the value 3 to a variable named "second" and output the contents of the variable.

```r
second<-3
second
```

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

The basic types for a single value are logical, numeric, and character. The same variable can hold values of multiple types in succession.

Note that a variable can hold a string of characters, as in the "Hello, World" example.

The str function is useful for getting detailed information about the structure of a variable in compact form.

```
y<-TRUE # logical or boolean value of TRUE
str(y)
```

```
##  logi TRUE
```

```
Y<-"TRUE" # character value of "TRUE"
str(Y)
```

```
##  chr "TRUE"
```

```
Y<-77

# The function "str" gives the structure of the argument.
# Each line of comments needs a "#" at the front.
str(Y)
```

```
##  num 77
```

In the code block below, please assign a logical value of FALSE to a variable named "second". Output the contents of the variable. Output the result of applying the function "str" to the variable "second". The result should show that "second" holds a logical value, not a character value.

```
second<-FALSE
str(second)
```

```
##  logi FALSE
```

Expressions

The usual arithmetic operations and the usual logical operations are available. Let's hold off for a moment on operations on character strings.

When control reaches an expression, that expression is evaluated and may be assigned to a variable. If it is not assigned, it will be output to the console (except in loops- more later). When

```
x<-5
y<-10
x+y
```

```
## [1] 15
```

```
add.em<-x+y
add.em
```

```
## [1] 15
```

Assignment of one variable to another copies the value on the right into the variable on the receiving end of the assignment. The two variables are fully independent.

```
add.copy<-add.em
add.copy<-10*add.copy
add.copy
```

```
## [1] 150
```

```r
add.em
```

```
## [1] 15
```

A variable may appear on both sides of the assignment. The right-hand side is computed, then assigned. There is no "x++" or "++x" incrementation syntax.

```r
x
```

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

```r
x<-x+1
x
```

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

In the code block below, please create a variable named "z" assigned the numeric value 5. Raise "z" to the power 3 using the syntax "z^3". Assign the result to "z" and output the new value of "z".

```r
z<-5
z<-z^3
z
```

```
## [1] 125
```

Vector Variables

Multiple values of the same type may be stored in a vector variable.

To assign a collection of values to a variable as a vector, separate the values by "," and enclose the sequence in "c(" followed by ")".

```r
fib.vec<-c(0,1,1,2,3,5,8)

str(fib.vec)
```

```
##  num [1:7] 0 1 1 2 3 5 8
```

```r
fib.vec
```

```
## [1] 0 1 1 2 3 5 8
```

```r
5*fib.vec
```

```
## [1]  0  5  5 10 15 25 40
```

```r
fib.vec+fib.vec
```

```
## [1]  0  2  2  4  6 10 16
```

```r
fib.vec^2
```

```
## [1]  0  1  1  4  9 25 64
```

```r
var.names<-c("age","income","gender")
var.names
```

```
## [1] "age"    "income" "gender"
```

```r
str(var.names)
```

```
##  chr [1:3] "age" "income" "gender"
```

In the code block below, please create a vector "myvec" with the values 0,0.5,1,2,3. Show the results of raising 4 to each of these powers using the syntax "4^myvec".

```r
myvec<-c(0,0.5,1,2,3)
4^myvec
```

```
## [1]  1  2  4 16 64
```

Random values

R uses a pseudorandom number generator to generate terms that function as random values. For reproducibility, set a random seed to start the pseudorandom number generator in the same place every time. Watch what happens running forward from a set seed then resetting the seed.

```r
set.seed(6789098)
sample(fib.vec,3)
```

```
## [1] 1 8 3
```

```r
sample(fib.vec,3)
```

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

```r
set.seed(6789098)
sample(fib.vec,3) # Note this is the same sample as the first one above.
```

```
## [1] 1 8 3
```

if-statements

Code blocks are grouped using "{" and "}". The "else" or "else if" should be on the same line as the closing bracket of the previous statement. Please track the effects of each conditional block below.

```r
if(x<10){
  x<-11
  y<-7
}

if(x<10){
  x<-11
  y<-7
} else if(x<80) {
  x<-9080
} else {
  y<-44
}
```

and again:

```r
if(x<10){
  x<-11
  y<-7
} else if(x<80) {
  x<-9080
} else {
  y<-44
}
```

In the code block below, please set the variable "z" to equal the character string "hi", a variable "w" to equal the character string "there", and a variable "t" to "ans". Please write an "if-else" structure that tests if "Z" is greater than "w". If it is, set "t" to "hi greater than there". Use an "else" structure to set "t" to "hi is less than or equal to there" otherwise. Output "t". (The order is lexicographic.)

```
z<-"hi"
w<-"there"
t<-"ans"
if(z>w){
  t<-"hi greater than there"
} else {
  t<-"hi is less than or equal to there"
}
t
```

```
## [1] "hi is less than or equal to there"
```

for-statements

You can iterate over an index or over the content of a vector. Note that vector indexing starts at 1.

```
counter<-0
for ( i in 1:length(fib.vec)){
  print(fib.vec[i])
  counter<-counter+1
}
```

```
## [1] 0
## [1] 1
## [1] 1
## [1] 2
## [1] 3
## [1] 5
## [1] 8
```

```
for ( i in fib.vec){
  print(i)
  counter<-counter+1
}
```

```
## [1] 0
## [1] 1
## [1] 1
## [1] 2
## [1] 3
## [1] 5
## [1] 8
```

Output is suppressed within loops.

No output is generated, but "counter" increments from 14 to 21.

```
for ( i in fib.vec){
  i
  counter<-counter+1
}
```

In the code block below, please iterate through "fib.vec", outputting the current value times 7. Don't increment the variable counter.

```r
for ( i in fib.vec){
  print(i*7)
}
```

```
## [1] 0
## [1] 7
## [1] 7
## [1] 14
## [1] 21
## [1] 35
## [1] 56
```

while-statements

```r
i<-1
while (fib.vec[i]<5){
  print(fib.vec[i])
  i<-i+1
}
```

```
## [1] 0
## [1] 1
## [1] 1
## [1] 2
## [1] 3
```

In the code block below, please iterate backward through "fib.vec" from the last (7th) value toward the first, continuing while the current value is greater than 2.

```r
i<-7
while (fib.vec[i]>2){
  print(fib.vec[i])
  i<-i-1
}
```

```
## [1] 8
## [1] 5
## [1] 3
```