Psyc 2011, Jones

Summary of Lab Week 1

Basic calculator

Addition:

```
> 7+4
```

Subtraction:

```
> 7-4
```

Multiplication:

```
> 7*4
```

Division:

```
> 7/4
```

Exponents:

```
> 7^4
```

Modular arithmetic (remainders):

```
> 7%%4
```

Variables

A variable is a name you create that stands for some number or set of numbers. You can create a variable by assigning it a value:

```
> x = 3
```

You can also make a variable equal the result of a computation:

```
> x = 3*4
```

Note: The = sign doesn't mean equal; it means assign (or, make equal). It assigns whatever's on the right to be the value of the variable on the left. Some things that don't work:

```
> 4 = x
> x + y = 7
```

You can find the current value of a variable by entering its name:

```
> x
```

You can use variables in calculations just like they were numbers:

```
> x * 7
```

Variable names must start with a letter and can only contain letters, numbers, periods, and underscores. Examples of allowable variable names:

```
 x, A, my_Variable, my.other.variable, variable3
```

Examples of unallowable variable names:

```
 3x, my#variable, v@riable
```

Variable names are case-sensitive, so you could have variables called `x` and `X`, and they would be distinct.

Functions

A function computes some function of the values you give to it. The function and its name already exist within R. The way you write it is `functionName(input1, input2, ...)`.

The `sum()` function adds up the values you give it:

```
> sum(3,5,6)
> sum(x,4,y,x)
```

The `prod()` function does products:

```
> prod(3,4,5)
```

The `log()` function does natural (base e) logarithms:

```
> log(3)
```

If you give `log()` two entries instead of one, it uses the second as the base.

```
> log(100,10)
```

The `exp()` function does exponentiation:

```
> exp(x)
```

This will return `e^x` (where e = 2.71828…).

You can combine functions with other operations and assignments.

```
> z = sum(2, 3*5, 7) * 2 + prod(3,4)
```

In this example, first `sum(2, 3*5, 7)` is evaluated, then the result (`29`) is multiplied by `2`, then `prod(3,4)` is evaluated and added to the first part, and then the result of the whole computation is assigned to the variable `z`.

If you give a function more (or fewer) entries than it can handle, it objects. These commands will give errors:

```
> exp(3,5)
> log(2,2,2)
```

Notice that `sum()` and `prod()` can take as many entries as you give them.

Vectors

In statistics we use vectors to represent sets of measurements (e.g. a score for each person). Vectors are created in R using the special `c()` function, which means concatenate. `c()` takes a set of numbers or vectors and concatenates them into one vector. `c()` isn't a function in the sense that it computes something, but it is a function in the R sense, because it takes a set of inputs and turns them into a specific output.

```
> X = c(2,4,5,6,3)
> X
[1] 2 4 5 6 3
```

You can also concatenate whole vectors:

```
> c(X,12,X)
[1]  2  4  5  6  3 12  2  4  5  6  3
```

Components of vectors

Each entry in a vector is called a component. If you want to see just one or few components of a vector, you use square brackets:

```
> X = c(2,4,5,6,3)
> X[5]
[1] 3
```

You can do arithmetic with the components of a vector:

```
> X[3] + 1
[1] 6
> X[3]*x[4]
[1] 30
```

The entry inside the brackets is called the index. You can use a vector for the index to get multiple components.

```
> Y = c(1,2,3)
> X[Y]
[1] 2 4 5
```

Note: The output is always in the same order as your index.

```
> X[c(5,3,1)]
[1] 3 5 2
```

Arithmetic with vectors

A scalar is a single number (i.e., not a vector). Adding or multiplying a vector by a scalar applies that operation to every component of your vector.

```
> x + 2
> x*3
```

You would do this if you had to transform a set of data from one measurement scale to another

```
> fah = cel * 9/5 + 32
> min = sec/60
```

Adding or multiplying two vectors is done component-by-component. You would do this if you needed to combine two variables.

```
> exam1 = c(87,83,66,97)
> exam2 = c(89,90,87,78)
> exam1 + exam2
[1] 176 173 153 175
```

```
> daysWorked = c(3,6,5,8,7)
> hoursPerDay = c(8,4,6,6,8)
> daysWorked * hoursPerDay
[1] 24 24 30 48 56
```

The : operator

Often you want a vector of the form `c(1,2,3,4,5,...)`. The `:` operator does this.

```
> 1:5
[1] 1 2 3 4 5
```

```
> x = 7
> 1:x
[1] 1 2 3 4 5 6 7
```

It can start and end anywhere

```
> -7:-2
[1] -7 -6 -5 -4 -3 -2
```

It can have a fractional part

```
> .5:6.5
[1] .5 1.5 2.5 3.5 4.5 5.5 6.5
```

Truth values

If you input a statement that can be either true or false, R gives you a result of `TRUE` or `FALSE`. `TRUE` and `FALSE` are not variables because you can't define them; it's best to think of them as special (logical) numbers.

```
> 1 < 2
[1] TRUE
> 2 > 7
[1] FALSE
> 2*6 > 9-4
[1] TRUE
```

If you want to evaluate an equality, i.e. a statement that two things are equal, use `==`. (Remember, single `=` means assign, not equals.)

```
> 1 == 1
[1] TRUE
> 2*3 == 6
[1] TRUE
> 1+1 == 3
[1] FALSE
```

If one side of your statement is a vector, R evaluates the equation for every component, and returns a vector of `TRUE`s and `FALSE`s.

```
> examScore = c(92,86,98,75)
> cutoff = 90
> examScore > cutoff
[1]  TRUE FALSE  TRUE FALSE
```

If both sides are vectors, then component 1 on the right side is compared to component 1 on the left side, and so on.

```
> preTest = c(92,79,81,89)
> postTest = c(90,85,81,93)
> postTest > preTest
[1] FALSE  TRUE FALSE  TRUE
```

```
> postTest == pretest
[1] FALSE FALSE  TRUE FALSE
```

A statement comparing vectors must use vectors of the same length. This will give an error:

```
> X = c(4,7,6)
> Y = c(4,8,6,7,3)
> X == Y
```

Using `TRUE` and `FALSE` as indices

An input like `X[c(1,4,5)]` means give me the 1st, 4th, and 5th components of `X`.
Another way to select components of a vector is with a list of `TRUE`s and `FALSE`s. This tells R to give us the components that correspond to `TRUE`s but to skip ones corresponding to `FALSE`s.

```
> X = c(3,6,5,8,6)
> Y = c(TRUE,FALSE,FALSE,TRUE,TRUE)
> X[Y]
[1] 3 8 6
```

This is useful for selecting a subset of your data that meets some criterion:

```
> examScore[heightInches > 72]
```

In this example, the expression `heightInches > 72` is computed first and results in a truth vector with one entry (`TRUE` or `FALSE`) for every subject. This truth vector is then used as the index for `examScore`. R outputs the values of `examScore` for which `heightInches > 72` is `TRUE`.

Strings

A string is non-numeric information, like a label.

```
> "glorp"
[1] "glorp"
> x = "glorp"
> y = "yingle"
> c(x,y)
[1] "glorp"  "yingle"
```

We use strings for nominal variables.

```
> sex = c("male","male","female","female","male")
```

You can't do much with string variables, but you can get truth values.

```
> sex == "male"
[1]  TRUE  TRUE FALSE FALSE  TRUE
```

We can then use these truth values to select subsets of data on other variables:

```
> examScore = c(87,68,96,57,82)
> examScore[sex=="male"]
[1] 87 68 82
> examScore[sex=="female"]
[1] 96 57
```