A short introduction to R

This handout provides a very basic introduction to R for the purpose of **Escape Excel** workshop. This handout will teach you the very basics, but not a robust introduction to **R**. For this, you should continue with other tutorials:

- Start with [https://www.statmethods.net/r-tutorial/index.html]
- Continue with: [http://tryr.codeschool.com]
- Advanced R: [http://adv-r.had.co.nz/]

What will you need

To use **R**, you need to download it in the first place. Go to [https://cran.r-project.org/] and follow instructions.

The other thing that you would need is just text editor or integrated development environment (IDE). Any text editor will suffice. Most modern text editors have code highlighting and some have even modules that turn them into mini-IDE, just faster and leaner. *Vim*, *gedit*, *Atom* and *Notepad++* are all good choices. IDEs have usually more programming and language-specific functions, but you won't generally need them (R is not Java:)). Many people are starting with *Rstudio*, a custom-build IDE for R.

Introduction to programming

In most programming languages, you will meet with three basic types of stuff (I don't want to say object, since they have specific meaning), **variables**, **functions** and **control flow structures**.

Variables

Variables are containers that hold some values or objects. You can assign to them by using either `=` or `<-` operators. While there exists minor differences between them, it is mostly of historical relevance. There are three main types of variables: **vectors**, **matrices** and **data.frames**.

Vectors

Vectors can be either **numeric** for numbers, **characters** for text and **lists** which are special types that can store essentially anything. You can create vector with `c()` and list with `list()`:

```
# this is comment and won't be interpreted
numeric_variable = 5
character_variable = "Hi, how are you today?"

numeric_vector = c(1, 3, 8)
numeric_series = 2:7
character_vector = c("Hi", "how", "are", "you?")

list = list(3, "Hi", 2:4)
```

Lists can be nested, which enable creation of complex structures:

```
vehicles = list(
    "cars" = c("fiat", "volvo", "mazda"),
    "planes" = c("boing", "airbus"),
    "motorcycles" = list(
        "sport" = c("kawasaki"),
        "choppers" = c("harley")
        )
    )
```

You can subset these structures to get single item or subvector. Vectors are subsetted using single `[`, for lists, you need to use two `[[`:

```
character_vector[c(2,4)]
```

```
## [1] "how"  "you?"
```

Vectors and lists can be named (as we did with lists). This allows you to subset using the name:

```
vehicles[["planes"]]
```

```
## [1] "boing"  "airbus"
```

Additionally, you can subset named lists with `$`, this makes lists very convenient containers.

```
vehicles$planes
```

```
## [1] "boing"  "airbus"
```

You can also use subsetting to modify specific vector:

```
character_vector[1] = "Cheers"
character_vector
```

```
## [1] "Cheers" "how"    "are"    "you?"
```

Matrices

Matrices are two-dimensional vectors. You can create matrix with function `matrix(data, rows, columns)`:

```
mat = matrix(5:8, 2, 2)
mat
```

```
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
```

Since matrix is two-dimensional, you need to specify both indexes while subsetting.

```
mat[1, ] # shorthand for first row
```

```
## [1] 5 7
```

Data frames

data.frames are lists that behave like matrices, which makes them very convenient for data analysis as they can contain varied data types. Such is their convenience that many other languages, such as Python, are implementing their own data.frames in their data analysis packages. Note that since they behave like matrices, they need to have to be rectangular.

```
table = data.frame(
        "cars" = c("fiat", "volvo"),
        "planes" = c("boing", "airbus"),
        "motorcycles" = c("kawasaki", "harley")
        )
table
```

```
##    cars planes motorcycles
## 1  fiat  boing    kawasaki
## 2 volvo airbus      harley
```

You can subset data.frames as you subset matrix. Additionally, you can use `$` as you did with lists:

```
table$planes
```

```
## [1] boing  airbus
## Levels: airbus boing
```

Functions

While you will mostly use already predefined functions, defining your own is a critical part of making your code more efficient.

```
function_name = function(param1, param2){
    # function body
    return(value) # return value
    }

output = function_name(par1, par2)
```

For example, lets define function `add` that will add together two parameters `x` and `y`. There is several ways how you can do that:

```
add = function(x, y){
    x + y # implicit return
    }


add = function(x, y){
    res = x + y
    res # implicit return of stored variable
    }


add = function(x, y){
    return(x + y) # explicit return
    }


add = function(x, y){
    print(x + y) # just prints without returning
    }
```

The particular style that you will use depends on your preferences and requirement in readability or structure.

Control flow structures

For most time, you will need to use two control flow structures: **for loops** and **conditional statements**.

for loops

Often, you need to repeat certain task several times. Such as calculate some equation for several specific variables. This can be easily done with for loops:

```
vector = 1:5
for(counter in vector){
    print(counter)
    }
```

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

Conditional statements

Other times, value that you want to return depends on some condition. This can be done with if-then constructs:

```
if(condition){
    # do something
    } else {
    # do something else
    }
```

Note that the else construct is optional. For example, lets create function that will process our condition and print `YES` if condition is `TRUE` and `NO` otherwise:

```
isItGood = function(condition){
    if(condition){
        print("YES")
        } else {
        print("NO")
        }
    }
isItGood(TRUE)
```

```
## [1] "YES"
```

```
isItGood(FALSE)
```

```
## [1] "NO"
```

```
isItGood(3+5 == 4)
```

```
## [1] "NO"
```

Documentation, packages and loading code.

So called "batteries" are already included in R. This means that by default, R has plenty of tools to process data, do analysis and make images. You usually use *google* to find what you want, but once you know the function that you are searching for, you can easily get documentation by writing `?function` or `help(function)`. Additionally, you can search inside all documentations with `??string`, which will find occurrence of `string` in documentation to functions and thus help you with discovery of tools in R.

A lot of times, you need to do some specialized thing. The popularity of R means that there are a LOT of packages doing just your thing, often the newest statistical methods are implemented in R first. To get them, you first need to know the package name. But then, installing it is easy:

```
install.packages(package)
```

and then you can load it an any time with:

```
library(package)
```

Quite often, you will have a small script where you will save your code. While you can copy and paste its content (and you will do it during debugging), it is often better to load all the code at once with `source(file)`. This will automatically run the code in the script as well.

Closing thoughts

This is the end of this simple introduction. There is still much more to learn, such as how to make pretty figures or specific domain knowledge, such as how to process DNA data or time-series. If you are just starting, you will almost immediately forget a lot of things you have just learned. For this reason, practicing your newly-gained knowledge is important, for this are ideals various interactive R tutorials. Once you feel comfortable in R, you can start reading other people's solution and learning from that. There are numerous portals where you can try to solve problem and when you do, you have access to number of other solutions. There you can often discover that there is not just a single way to do things in R, but multiple approaches, each with its advantages and disadvantage.