PROGRAMMING WITH R

Functions, if then statement, loops

Why do programming?

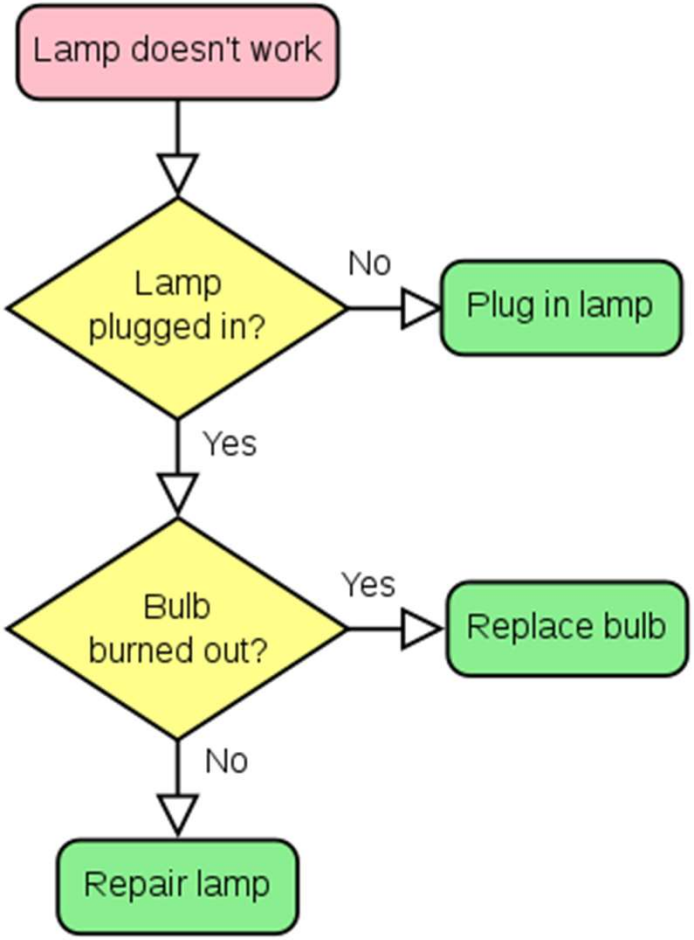
- If you use other people's software, you will always be limited by what other people think you want to do.
- Write your own programs and the only limit will be your own imagination.
- Even with all those programs on your computer, you still need to do something different, something specific to you.

A few basic things first…

Flowchart

- Programming is the process of writing instructions in a language that the computer understands.
- In many ways, flowcharts are useful to see/visualize how a program works.
- When writing codes, we must think first of what we want and how we want to construct it.

Flowchart


Lamp doesn't work
Lamp plugged in?
No
Plug in lamp
Yes
Bulb burned out?
Yes
Replace bulb
No
Repair lamp

Printing results/values

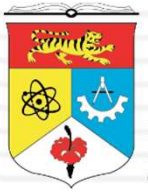
- The function `print()` prints out the value inside the bracket.
- The function `paste()` allows to concatenate/combine strings.
- Example:

```
> x <- 10
> print(x)
[1] 10
> print(paste("The value of x is", x))
[1] "The value of x is 10"
```

Comments

- To write comments in R, we use the # character.
- R will ignore anything written in the line after the # character.
- Example:

```
> x <- 10        # value of x
> y <- 10        # value of y
> z <- x + y     # sum of x and y
```



UNIVERSITI KEBANGSAAN MALAYSIA
The National University of Malaysia

Functions

Functions

- We have used functions extensively so far. For example,
  - `sqrt()`
  - `getwd()`
  - `solve()`
  - `sum()`

- A function is a block of organized, reusable code that is used to perform a single, related action.

Functions

- Every function in R has three basic parts:
  - name
  - set of arguments
  - body of code

- The following is the basic construction of a function:

```
function_name <- function(argument1,argument2,...){
    //codes
    return(output)
}
```

Functions

- Then, you can use the function by running
  - `function_name(value1, value2, ...)`, or
  - `function_name(argument_1 = value1, ...)`, or
  - `function_name()`, if the function has no argument.

- Note that the `return(output)` part of the code is not necessary but is recommended to be added.

- Without it, the last line read by the function is returned instead.

- The function will stop when it reaches the `return(output)` statement, without reading the following lines.

Functions

- Example of a function that squares its argument:

```
f <- function(x){
     y <- x*x
     return(y)
}
```

```
> f(2)
[1] 4
> f(x=3)
[1] 9
> z = f(4)
> z
[1] 16
```

Functions

- Example of a function that calculates x^y:

```
x_y <- function(x, y){
    return(x^y)
}
```

```
> x_y(3,2)
[1] 9
> x_y(2,5)
[1] 32
> x_y(x=3, y=5)
[1] 243
> x_y(y=5, x=3)
[1] 243
```

Setting default values for the arguments

- You can set the default value for an argument of a function by using `argument = value` when writing the function.

- Eg:

```
function_name <- function(argument_1 = value,
                          argument_2 = value, ...){
    //codes
    return(output)
}
```

Setting default values for the arguments

- Example:

```
f <- function(x = 10){
     y <- x*x
     return(y)
}
```

```
> f()
[1] 100
> f(3)
[1] 9
```

Setting default values for the arguments

- Example:

```
x_y <- function(x, y = 2){
    return(x^y)
}
```

```
> x_y(4)
[1] 16
> x_y(4,3)
[1] 64
```

Multiple outputs

- If there are multiple outputs in the function, we can return list instead. Lists are just "collections of variables/objects".
- Dollar sign `$` is used to extract the variable of a list.
- The function `names()` can be used to list down all variable names in the list.
- Syntax creating list:

```
list(var1_name = value, var2_name = value, ...)
```

Multiple outputs

- Example:

```
f <- function(x){
  x1 <- x
  x2 <- x^2
  x3 <- x^3
  return(list(x1 = x1, x2 = x2, x3 = x3))
}
```

```
> y <- f(2)
> y$x1
[1] 2
> y$x2
[1] 4
> y$x3
[1] 8
> names(y)
[1] "x1" "x2" "x3"
```

Additional notes on functions

- Objects that are created within the function are local to the environment of the function. They don't exist outside of the function.

- The argument can be any type of object (scalar, matrix, vector, data frame or logical).

Good function writing practices

- Before writing your codes, take note first of what your input is, and what you want the output to be.
- Keep your functions short.
  - If things start to get long, you can probably split up the function into several functions.
  - It also makes your code easier to update.
- Put in comments on what are the inputs, what the function does and what is the output.
- Check for errors along the way.

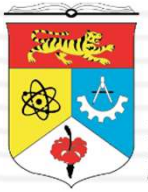
Exercises

1. Write a function named `abs_diff` that takes two values and calculates the absolute value of the difference between two values, i.e., $|x - y|$.

2. The Euclidean distance between two n-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ is

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Write a function named `euc_dist` that calculates the Euclidean distance between two vectors.



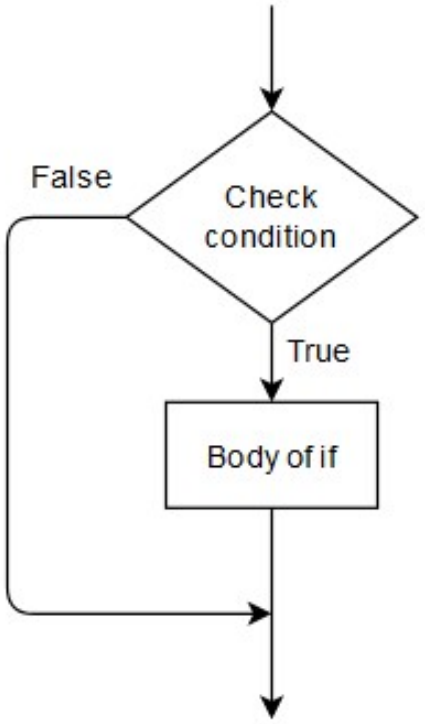
If else statements and loops

If statement

- What it is for:
  - Want to run a block of code ONLY if a condition is met.

- Syntax:

```
if (condition) {
    //code executed if
      condition is True
}
```



If statement

- Example:

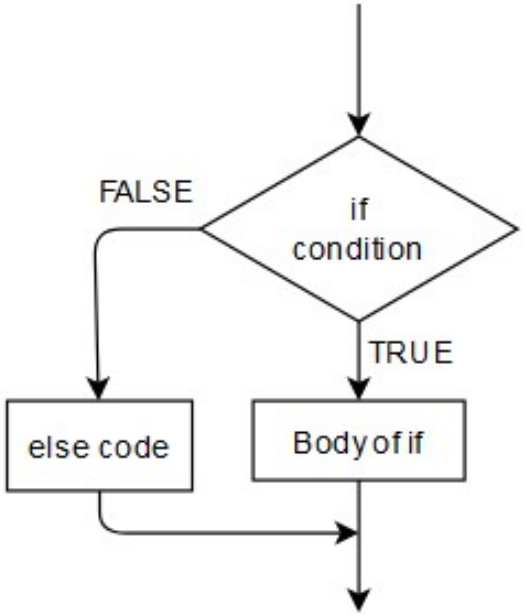
```
> x <- 5
> if (x < 10) {
+      print("x < 10")
+ }
[1] "x < 10"
>
> x <- 35
> if (x < 10) {
+      print("x < 10")
+ }
>
```

If else statement

- What it is for:
  - Want to run a block of code if a condition is TRUE.
  - Otherwise, run another block of code.

- Syntax:

```
if (condition) {
  //code executed if condition
    is True
} else {
  //code executed if condition
    is False
}
```



If else statement

- Example:

```
> x <- 5
> if (x<10) {
+     print("x < 10")
+ } else {
+     print("x >= 10")
+ }
[1] "x < 10"
>
> x <- 13
> if (x<10) {
+     print("x < 10")
+ } else {
+     print("x >= 10")
+ }
[1] "x >= 10"
```

If else if statement

- Similar to if else statement, but with more conditions.

- Syntax:

```
if (condition1) {
    //code executed if condition1 is True
  } else if (condition2) {
    //code executed if condition2 is True
  } else {
    //code executed if condition1 and
      condition2 are False
}
```

If else if statement

- Example:

```
f <- function(x){
    if (x<10) {
        print("x < 10")
    } else if (x>=10 & x < 20){
        print("10 <= x < 10")
    } else {
        print("x >= 20")
    }
}
```

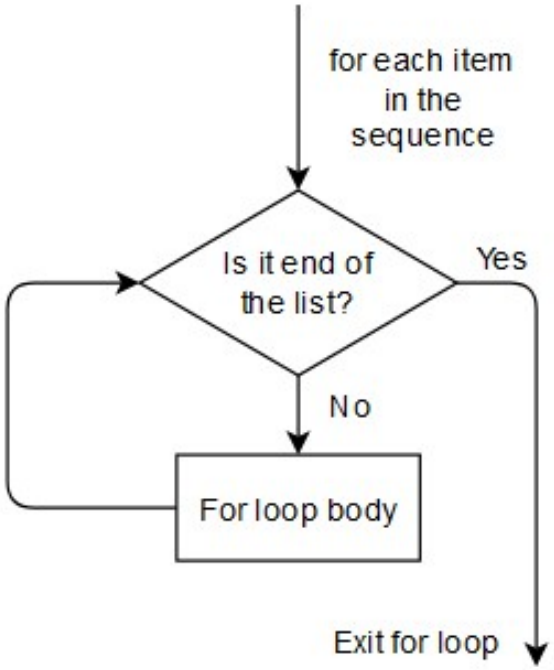
```
> f(5)
[1] "x < 10"
> f(13)
[1] "10 <= x < 10"
> f(23)
[1] "x >= 20"
```

For loop

- What it is for:
  - Want to repeat codes but for different value of i (or any other variable name)
  - The values of i is specified in a vector

- Syntax:

```
for(i in vector){
    //body
}
```



For loop

- Example:

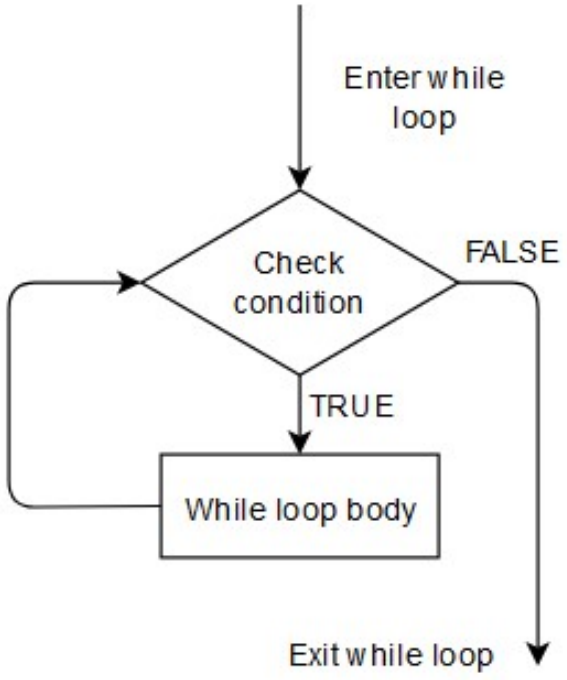
```
> x <- 0
> for(i in 1:10){
+       x = x + i
+ }
> x
[1] 55
>
> for(j in c(1,5,2)){
+       print(j)
+ }
[1] 1
[1] 5
[1] 2
```

While loop

- What it is for:
  - Want to repeat codes UNTIL the condition is no longer met.

- Syntax:

```
while(condition){
    \\code executed if
      condition is True
    \\stop loop if condition
      is False
}
```



While loop

- Example:

```
> x <- 2
> i <- 0
> while(x<=10){
+       x = 2*x
+       i = i+1
+ }
> x
[1] 16
> i
[1] 3
```

Breaking the loop

- We can also force for or while loops to stop by using `break`.

- Example:

```
> for(val in 1:10){
+     print(val)
+     if(val == 4){
+         break
+     }
+ }
[1] 1
[1] 2
[1] 3
[1] 4
```

Exercises

1. Without using `max` function, write a function in R that takes two values and will output the maximum of the two values. Additionally, the function will display a message if the two values are equal.

2. Without using the `sum` function, write a function in R that calculates the total summation of a vector that
   a) uses the for loop.
   b) uses the while loop.