Which expression(s) equal to TRUE? (`x` equals 5)

a) `TRUE | FALSE`
b) `x > 5`
c) `FALSE & TRUE`
d) `x <= 10 | x > 5`

Answer: a) and d)

What is the value of `y` at the end of the loop if it was 0 and the beginning? How many iterations of the loop occurred?

```
while (y <= 10) { y <- 2*y + 3}
```

Answer: y = 21; the loop ran 3 times.

An introduction to R
WS 2019/2020

Basics of Algorithmics in R

Dr. Noémie Becker
Dr. Eliza Argyridou

Special thanks to:
Dr. Sonja Grath for addition to slides

What you should know after days 7 & 8

Review: Data frames and import your data

Conditional execution in R

- Logic rules
- if(), else(), ifelse()
- Example from day 1

Loops

Executing a command from a script

Writing your own functions

How to avoid slow R code

Basics

Syntax:

```
myfun <- function (arg1, arg2, …) { commands }
```

Example:
We want to define a function that takes a DNA sequence as input and gives as output the GC content (proportion of G and C in the sequence).

How can we name our function?
Idea: `gc`

```
?gc
# There is already a function gc()
```

Another idea: `gcContent`

```
?gcContent
No documentation for 'gcContent' in specified packages
and libraries:
you could try '??gcContent'
```

→ We can name our function `gcContent()`

Our function gcContent() from Day 1

Version 1

```
gcContent <- function(dna, counter = 0){
    dna <- unlist(strsplit(dna, ""))
    for(i in 1:length(dna)){
        if(dna[i] == "C" | dna[i] == "G")
            {counter = counter + 1}
    }
    return(counter/length(dna))
}
```

Does our function works correctly?


```
# Test the function with some example data
gcContent("AACGTGGCTA")
gcContent("AATATATTAT")
gcContent(23)
gcContent(TRUE)
gcContent("notDNA")
gcContent("Cool")
```


Dealing with problems

Problems:

- R gives an error message if the input is not a character value
- Our function calculates values if the input is most likely not a DNA sequence

How could we deal with these problems?

What do we want our function to output in these cases?

Error and Warning

There are two types of error messages in R:

- **Error**: Stops execution and returns no value
- **Warning** message: Continues execution

Example:

```
x <- sum("hello")
Error in sum("hello") : invalid 'type' (character) of
argument

x <- mean("hello")
Warning message:
In mean.default("hello") :
  argument is not numeric or logical: returning NA
```

We can define such messages with the functions `stop()` and `warning()`

In our example:

- (Specific) Error when argument is not character
- Warning if character argument is not DNA

Dealing with non-character arguments

Version 2:

Self-defined error message

```
gcContent <- function(dna, counter = 0){
    if (!is.character(dna)){
      stop("The argument must be of type character.")
    }
    dna <- unlist(strsplit(dna, ""))
    for(i in 1:length(dna)){
        if(dna[i] == "C" | dna[i] == "G")
        {counter = counter + 1}
    }
    return(counter/length(dna))
}
```


Dealing with input that is not DNA

- We define as 'not DNA' any character different from A, C, G or T.
- If the input contains any other character, we compute the value but throw a warning.

To solve this task, we can use the function `grep()` as follows:

```
grep("[^ACGT]", "AATGAC")
Integer(0)                  # length is 0
grep("[^ACGT]", "NATGAC")
[1] 1                       # length is 1
```


Dealing with input that is not DNA

Version 3

```
gcContent <- function(dna, counter = 0){
    if (!is.character(dna)){
        stop("The argument must be of type character.")
    }
    if (length(grep("[^ACGT]", dna)) > 0){
        warning("The input contains characters other than A,
        C, G or T - value should not be trusted!")
    }
    dna <- unlist(strsplit(dna, ""))
    for(i in 1:length(dna)){
        if(dna[i] == "C" | dna[i] == "G")
        {counter = counter + 1}
    }
    return(counter/length(dna))
}
```

Self-defined warning message

Giving several arguments to a function

R functions can have several arguments.
You can see them listed in the help page for the function.

Example

```
?mean()
```

A frequent argument in R functions is na.rm. This argument (when set to TRUE) removes NA values from vectors.


```
mean(c(1, 2, NA))
[1] NA

mean(c(1, 2, NA), na.rm = TRUE)
[1] 1.5
```

We now want to give our function another argument to output the AT content instead of the GC content.

Giving several arguments to a function

Version 4

```
gcContent <- function(dna, counter = 0, AT){
    if (!is.character(dna)){
        stop("The argument must be of type character.")
    }
    if (length(grep("[^ACGT]", dna)) > 0){
        warning("The input contains characters other than A,
        C, G or T - value should not be trusted!")
    }
    dna <- unlist(strsplit(dna, ""))
    for(i in 1:length(dna)){
        if(dna[i] == "C" | dna[i] == "G")
        {counter = counter + 1}
    }
    if (AT == TRUE){
        return(1 - counter/length(dna))
    } else {
        return(counter/length(dna))
    }
}
```

Giving several arguments to a function

Test:

```
gcContent("AACGTGTTTA", AT = TRUE)
[1] 0.7
gcContent("AACGTGTTTA")
Error in gcContent("AACGTGTTTA") :
  argument "AT" is missing, with no default
```

We should give the value `FALSE` to the argument AT per default and the argument would only be changed if the user specifies `AT = TRUE`

Calculating GC/AT content – Final version

```
gcContent <- function(dna, counter = 0, AT = FALSE){
    if (!is.character(dna)){
        stop("The argument must be of type character.")
    }
    if (length(grep("[^ACGT]", dna)) > 0){
        warning("The input contains characters other than A,
        C, G or T - value should not be trusted!")
    }
    dna <- unlist(strsplit(dna, ""))
    for(i in 1:length(dna)){
        if(dna[i] == "C" | dna[i] == "G")
        {counter = counter + 1}
    }
    if (AT == TRUE){
        return(1 - counter/length(dna))
    } else {
        return(counter/length(dna))
    }
}
```


Returning several values

If we want a function that returns several values (not only one), we can output a vector or a list.

Example:
We define a function `minmax()` that determines the minimum and maximum of a given vector and returns these two values.

```
minmax <- function(x){
    return(list(minimum = min(x),
                maximum = max(x)))
}
```

Test:

```
x <- c(1, 23, 2, 7, 0)
minmax(x)
$minimum
[1] 0

$maximum
[1] 23
```


Returning several values

What type of data does our function return?
We can assign the result to a variable to investigate this question further.

```
myResult <- minmax(x)
typeof(myResult)
[1] "list"
str(myResult)
List of 2
 $ min: num 0
 $ max: num 23
```

How can we access the individual values of a list?

```
myResult[[1]]
[1] 0
myResult[[2]]
[1] 23
myResult$minimum
[1] 0
myResult$maximum
[1] 23
```


Remember Lecture 3 – Data types and Structures

Almost all functions (e.g., t-test, linear regression, etc.) in R produce output that is stored in a list ...

→ We now have everything we need to know to access particular parts of such an output ☺

Example: heartbeat.txt (exercise sheet 7)

- We want to test for a difference in mean of weight increase for the two groups.
- The groups are composed of different individuals, therefore we can apply an unpaired *t*-test (more on *t*-tests tomorrow).

```
# Import the data
heartbeats <- read.table("heartbeats.txt", header = TRUE)
# Perform an unpaired t-test
t.test(heartbeats$wghtincr[heartbeats$treatment == 0],
       heartbeats$wghtincr[heartbeats$treatment == 1])
```

```
    Welch Two Sample t-test

data:  heartbeats$wghtincr[heartbeats$treatment == 0] and
heartbeats$wghtincr[heartbeats$treatment == 1]
t = -6.9307, df = 206.3, p-value = 5.254e-11
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -88.91976 -49.53478
sample estimates:
mean of x mean of y
-31.72727  37.50000
```

Example: heartbeat.txt (exercise sheet 7)

```
# Perform an unpaired t-test and save the result into a
# variable to investigate it further
t_test_result <-
t.test(heartbeats$wghtincr[heartbeats$treatment == 0],
       heartbeats$wghtincr[heartbeats$treatment == 1])

typeof(t_test_result)
[1] "list"
str(t_test_result)
List of 9
 $ statistic  : Named num -6.93
  ..- attr(*, "names")= chr "t"
 $ parameter  : Named num 206
  ..- attr(*, "names")= chr "df"
 $ p.value    : num 5.25e-11
 $ conf.int   : atomic [1:2] -88.9 -49.5
  ..- attr(*, "conf.level")= num 0.95
 $ estimate   : Named num [1:2] -31.7 37.5
  ..- attr(*, "names")= chr [1:2] "mean of x" "mean of y"
 $ null.value : Named num 0
  ..- attr(*, "names")= chr "difference in means"
 $ alternative: chr "two.sided"
 $ method     : chr "Welch Two Sample t-test"
 $ data.name  : chr "heartbeats$wghtincr[heartbeats$treatment == 0] and
heartbeats$wghtincr[heartbeats$treatment == 1]"
 - attr(*, "class")= chr "htest"
```

Imagine we are interested in just the p value.
How can we extract it?

```
t_test_result[[3]]
t_test_result$p.value
```


What you should know after days 7 & 8

Review: Data frames and import your data

Conditional execution in R

- Logic rules
- if(), else(), ifelse()
- Example from day 1

Loops

Executing a command from a script

Writing your own functions

How to avoid slow R code

How to avoid slow R code

- R has to interpret your commands each time you run a script and it takes time to determine the type of your variables.
- So avoid using loops and calling functions again and again if possible
- When you use loops, avoid increasing the size of an object (vector ...) at each iteration but rather define it with full size before.
- Think in whole objects such as vectors or lists and apply operations to the whole object instead of looping through all elements.

Take-home message

- Use the function `function()` to write your own functions in R.
- You can specify as many arguments as you need and you can choose their default value if needed.
- Use `warning()` and `stop()` for issuing error messages.
- Think about controlling the input from the user