Learning R with Kalekye

Kalekye

1. Signs

* Equals to (==)
* Not equal to (!=)
* Greater than (>)
* Less than (<)
* Greater than or equal to (>=)
* Less than or equal to (<=)

1. Operators AND (&) OR (|)
2. R if

* They are commonly used in if statements
* if statement is used to specify a block of code to be executed if a condition is TRUE
* {} are used to define the scope of a code/ function

a <- 30  
b <- 220  
  
if (b > a){  
 print("b is greater than a")  
}

## [1] "b is greater than a"

1. else if - you are telling R that if the previous conditions were not true, then try this other condition

a <- 30  
b <- 30  
  
if (b > a){  
 print("b is greater than a")  
} else if (a == b) {  
 print("b is equal to a")  
}

## [1] "b is equal to a"

1. if…else - capture anything not captured by if… and else if…

* if…else can work without else if

a <- 30  
b <- 220  
  
if (b < a) {  
 print("b is less than a")  
} else{  
 print("b is greater than a")  
}

## [1] "b is greater than a"

1. AND and OR operators - they are logical operators and are used to combine conditional statements

a <- 30  
b <- 220  
c <- 400  
  
if (b > a & c > a) {  
 print("both conditions are TRUE")  
}

## [1] "both conditions are TRUE"

a <- 30  
b <- 220  
c <- 400  
  
if (b < a | c > a) {  
 print("either conditions is TRUE")  
}

## [1] "either conditions is TRUE"

1. while loop - executes a statement as long as the condition is TRUE

* print i as long as i < 6

i <- 1  
while(i < 6){  
 print(i)  
 i <- i+1  
}

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

* we can combine while loop with if…else function
* print play a game football if the player number is 6

player <- 1  
while(player <= 6){  
 if (player < 6){  
 print("No football")  
 }else{  
 print("play a game football")  
 }  
 player <- player + 1  
}

## [1] "No football"  
## [1] "No football"  
## [1] "No football"  
## [1] "No football"  
## [1] "No football"  
## [1] "play a game football"

1. For loop - used to perform iteration

for(x in 1:10){  
 print(x)  
}

## [1] 1  
## [1] 2  
## [1] 3  
## [1] 4  
## [1] 5  
## [1] 6  
## [1] 7  
## [1] 8  
## [1] 9  
## [1] 10

* nest a if…else with a for loop

dice <- 1:10  
for(x in dice){  
 if (x == 6){  
 print(paste("The dice is", x, "football"))  
 } else{  
 print(paste("The dice is", x, "not football"))  
 }  
}

## [1] "The dice is 1 not football"  
## [1] "The dice is 2 not football"  
## [1] "The dice is 3 not football"  
## [1] "The dice is 4 not football"  
## [1] "The dice is 5 not football"  
## [1] "The dice is 6 football"  
## [1] "The dice is 7 not football"  
## [1] "The dice is 8 not football"  
## [1] "The dice is 9 not football"  
## [1] "The dice is 10 not football"

1. Functions - a block of code that runs when called

* We specify parameters (data)
* use function() to create

my\_function <- function(){  
 print("hello")  
}  
my\_function()

## [1] "hello"

* when calling a function, specify same number arguments as supplied

my\_function1 <- function(fname, mname ,lname){  
 paste(fname, mname, lname)  
}  
  
my\_function1("Joy", "Kalekye", "Joy")

## [1] "Joy Kalekye Joy"

* installing and using R packages
* for installation, you need internet
* install a package once

#install.packages("tidyverse")  
library(tidyverse)

## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──  
## ✔ dplyr 1.1.4 ✔ readr 2.1.5  
## ✔ forcats 1.0.0 ✔ stringr 1.5.1  
## ✔ ggplot2 3.5.1 ✔ tibble 3.2.1  
## ✔ lubridate 1.9.3 ✔ tidyr 1.3.1  
## ✔ purrr 1.0.2   
## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()  
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors