DSA1101

Introduction to Data Science

Week 3 Tutorial 1

HI, IM KAI RONG!

kaironglee@u.nus.edu | tele: @kaiironglee

about me:

- Year 3 Data Science and Analytics
- Took DSA1101 AY23/24 Semester 2
- Interest: XAI, Machine Learning, Data Visualization
- Workshop Head @ NUS Statistics and Data Science Society (SDS) & Curriculum Head @ NUS Product Club
- 1st time teaching DSA1101 (Ur feedback helps me improve!)
- Looking forward to learning together this semester

ABOUT TUTORIALS

[T5] Wednesday 12-2pm

- 5% of Final Grade
    - Valid reason (MC) for absence will not be penalised
- Each tutorial will be aimed to be 1.5 hours long (I'll try my best to end on-time)
- Feel free to stay by after the tutorial to ask me questions (abt the tutorial, data science, modules, uni society or anything!)

ABOUT TUTORIALS

1. Quick Revision
2. On-site Questions Attempt
3. On-site Questions Discussion
4. Off-site Questions Discussion

MY TUTORIAL STYLE

- I would try to make each tutorial as engaging as possible, so ur experience of learning data science will be very fun!

- I also value interactions! So if you can ask / answer at least one question each tutorial, I would be very happy 🤩 !

SOME ENCOURAGING TIPS

- I took DSA1101 with CS1010s in the same semester, so I had no experience in coding let alone in R before the module
- An important thing that rlly helped me for this module was **note-taking**, all my notes during lectures and tutorials are taken in R-studio only
- I also never used AI for this mod, which imo is the rlly important

```
######  TOPIC 2: BASIC PROBABILITY & STATISTICS
  ## there are two types of variable- quantitative and categorical
  ## quantitative is either discrete or continuous
  ## categorical is either
    ## ordinal(can ordered eg. education level) or
    ## nominal(cannot be ordered eg. race)

  ## Two ways to describe data: numerical, graphical
    ## numerical: number of observations, location, variability
    ## graphical: histogram, boxplot, QQ plot, scatter plot

setwd("/Users/kaironglee/Documents/R/data")
sales <- read.csv("yearly_sales.csv")

##### NUMERICAL SUMMARY OF DATA
  ## (for single variable)

head(sales)
total = round(sales$sales_total, digits = 2)
  ## total is all data of the sales_total
  ## take note of the the round
  ## $ means accessing
```

The bellcurve for this module is quite high, but I think if you really put in the effort to:

(1) go to every lecture and take note

(2) attempt tutorial questions

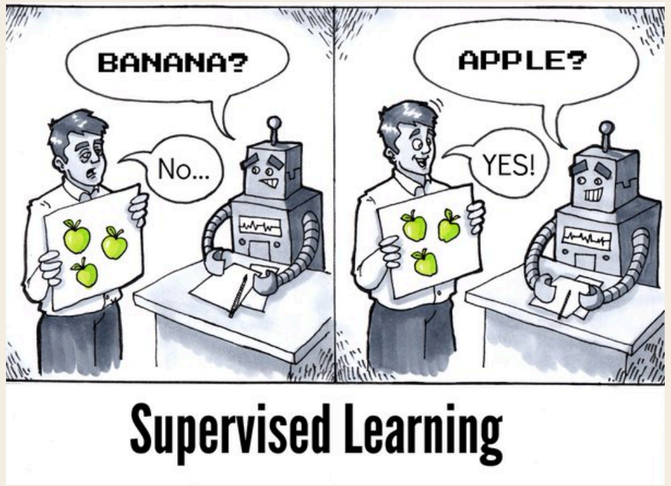
(3) put a good amount of effort into your project

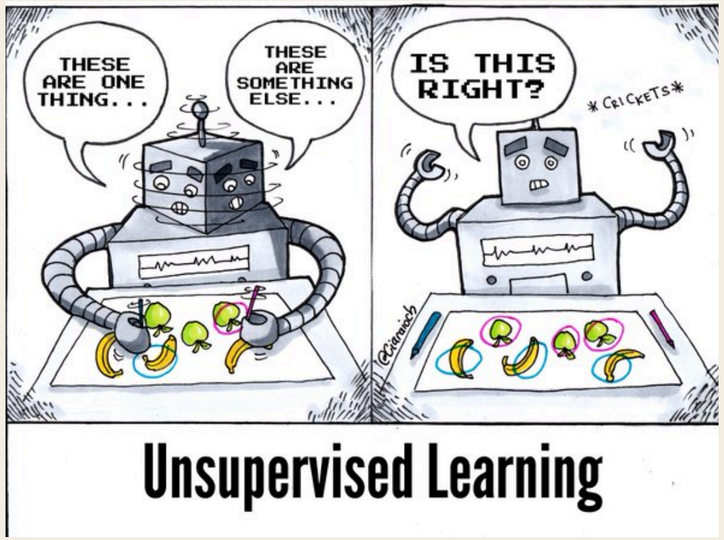
→ an A/A+ with no coding experience is definitely possible! 🤩

WHAT WILL YOU LEARN IN DSA1101?

All coding in this module will be in R!

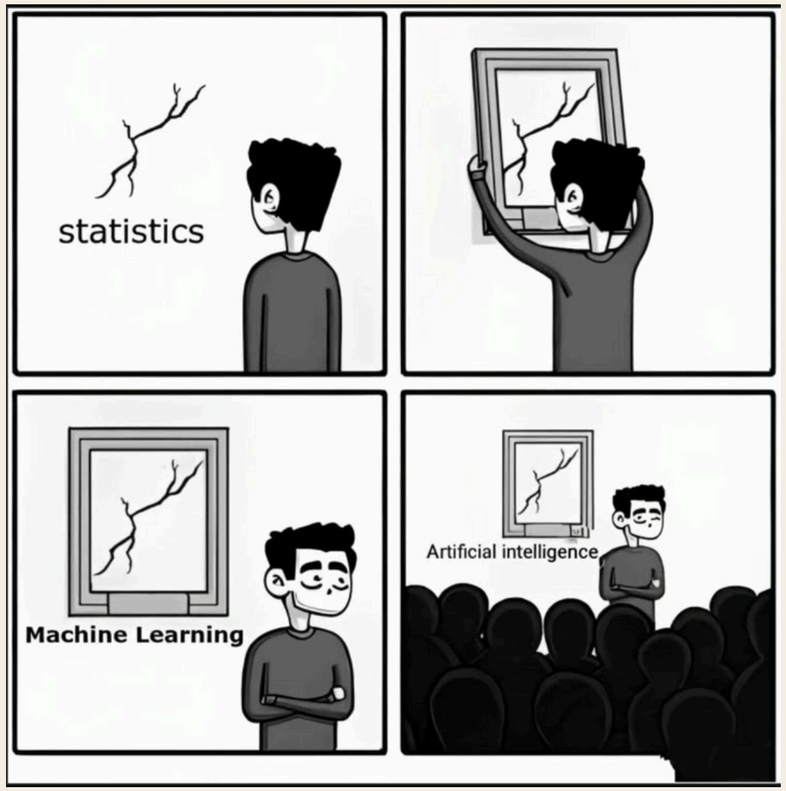
1. Introduction to R programming
2. Introduction to basic probability and statistics
3. Supervised learning
   - k-nearest neighbours
   - Decision trees
   - Regression analysis
   - Naïve Bayes
4. Diagnostics of classifiers
5. Model validation
6. Unsupervised learning
   - k-means clustering
   - Association rules





WHY R? WHY NOT PYTHON?

- People often ask why this module uses R, given how widely Python is used in industry (including Y1 me!)
- R is still commonly used in academia, pharma, biostats, and research, which aligns well with the statistical focus of this course
- Many ML and AI methods come from statistics, and R was designed specifically for **statistical modelling and analysis**
- R helps you focus more on understanding the models and the statistics, rather than dealing with programming complexity early on



PLENTY OF PYTHON AHEAD

- Plenty of Python opportunities in IT1244, CS3244 etc
- I'm also TAing IT1244 this sem, which implements a lot of the same models from this course but in Python (KNN, Linear Regression, K-Means)
- **If you're interested in the Python side: lmk and I can share Python implementations of what we're learning in this module! Happy to help you see both versions side-by-side**

QUICK REVISION

c() function

Example Code

```
vec <- c(1, 2)

combined_vec <- c(vec, 3)

print(combined_vec)
```

Output

```
[1] 1 2 3
```

`c()` combines/concatenate vectors.

It can also create vectors too!

c() function

Example Code

```
vec1 <- c(1, 2)

vec2 <- c(4, 5)

combined_vec <- c(vec1, vec2)

print(combined_vec)
```

Output

```
[1] 1 2 4 5
```

`c()` combines/concatenate vectors.

It can also create vectors too!

c() function

Example Code

```
vec1 <- c(1, 2)

vec2 <- c("a", "b")

combined_vec <- c(vec1, vec2)

print(combined_vec)
```

**When combining different data types in a vector, R converts everything to the most flexible type (eg. numbers become text because vectors can only hold one data type)

Output

```
[1] "1" "2" "a" "b"
```

c() function

Example Code

```
vec1 <- c(1, 2)

vec2 <- c(3, NA)

combined_vec <- c(vec1, vec2)

print(combined_vec)
```

NA preserves the vector's type when combined (any data type!).

Output

```
[1] 1 2 3 NA
```

append() FUNCTION

Example Code

```
vec <- c(1, 2)

combined_vec <- append(vec, 3)

print(combined_vec)
```

can also use append() to add elements in vectors!

Output

```
[1] 1 2 3
```

append() FUNCTION

Example Code

```
vec <- c(1, 2)

combined_vec <- append(vec, 3,
after = 1)

print(combined_vec)
```

you can also choose where in the vector you want to append your elements

Output

```
[1] 1 3 2
```

Indexing Elements

Example Code

```r
v <- c("a", "b", "c", "d", "e")

print(v[3])
```

Outputs third element of the vector:

index in R starts at 1, not 0 unlike Python

Output

```
[1] "c"
```

Indexing Elements

Example Code

```r
v <- c("a", "b", "c", "d", "e")

index <- c(1, 3, 5)

print(v[index])
```

Indexing multiple elements

Output

```
[1] "a" "c" "e"
```

Indexing Elements

Example Code

```
v <- c("a", "b", "c", "d", "e")

print(v[-3])
```

Negative indexing - excludes elements at specified positions from a vector.

Output

```
[1] "a" "b" "d" "e"
```

Indexing Elements

Example Code

```
v <- c("a", "b", "c", "d", "e")

index <- c(FALSE, TRUE, FALSE,
TRUE, TRUE)

print(v[index])
```

Logical indexing - selects elements where the corresponding logical value is TRUE.

Output

```
[1] "b" "d" "e"
```

Indexing Elements

Example Code

```
v <- c(1, 2, 3, 4, 5, 6)

print(v[v < 4])
```

Conditional indexing - selects elements that meet a specific condition.

Output

```
[1] 1 2 3
```

Indexing Elements

Example Code

```
v <- c("a", "b", "c", "d", "e")

print(v[10])
```

Out-of-bounds indexing - attempts to access an element at a position that doesn't exist.

Output

```
[1] NA
```

Indexing Elements

Example Code

```
m <- matrix(1:9, nrow = 3, ncol = 3)

print(m)

print(m[1,2])
```

Matrix indexing - accesses elements in a matrix using [row, column] notation.

Output

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9


[1] 4
```

IF ELSE STATEMENT

```
if (condition) {

  # code if TRUE

} else {

  # code if FALSE

}
```

IF ELSE STATEMENT

Example Code

```
age <- 20

if (age >= 18) {

  print("Adult")

} else {

  print("Minor")

}
```

Output

```
[1] "Adult"
```

Checks if age is 18 or older. If TRUE, prints "Adult"; if FALSE, prints "Minor"

LOOPS IN R

For Loop

```
for (variable in sequence) {

  # Code to execute

}
```

While Loop

```
while (condition) {

  # Code to execute

}
```

FOR LOOPS

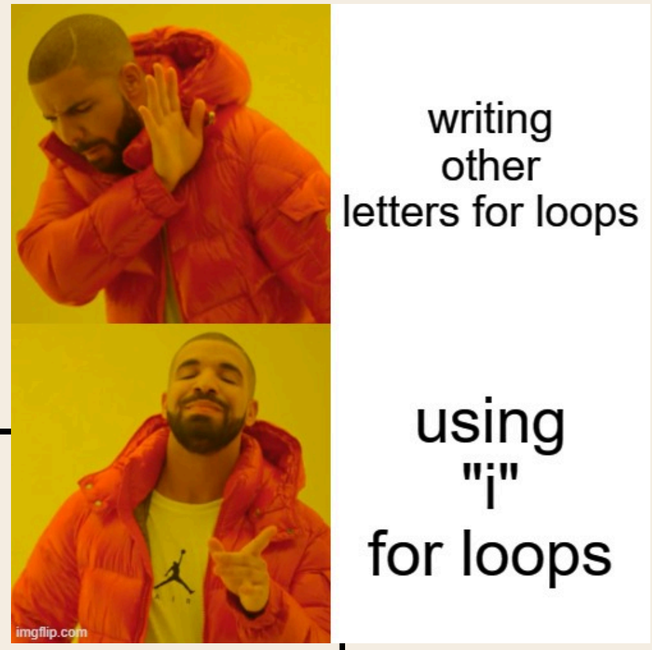
Example Code

```
for (i in 1:5) {

  print(i)

}
```

`print()` produces a new line of output

Output

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



WHILE LOOPS

Example Code

```
count = 1

while (count <= 5) {

  print(count)

  count <- count + 1
}
```

`print()` produces a new line of output

Output

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

which() FUNCTION

Example Code

```
data <- c(2, 4, 6, 8)

positions <- which(data > 5)

print(positions)
```

`which()` returns the positions (indices) where the condition is TRUE.

Output

```
[1] 3 4
```

How do I get the actual **values** (not positions) from data where data > 5?

which() FUNCTION

Example Code

```
data <- c(2, 4, 6, 8)

positions <- which(data > 5)

print(data[positions])
```

vector_name[which(condition)] - is how you can index the actual value from it’s position

Output

```
[1] 6 8
```

CREATE FUNCTION IN R

```r
function_name <- function(parameter1, parameter2 = default_value) {

  # code to execute

  return(result)

}
```

which() FUNCTION

Example Code

```r
multiply <- function(x, y = 2) {

  result <- x * y

  return(result)

}

print(multiply(5, 3))  # Uses both parameters
print(multiply(5))     # Uses default y = 2
```

Output

```
[1] 15
[1] 10
```

Creates a function that multiplies two numbers.

Parameter `y` has a default value of 2

%in%

Example Code

```
colors <- c("red", "blue", "green", "yellow")

print("blue" %in% colors)

print("purple" %in% colors)
```

Checks if a value exists in a vector. Returns TRUE if found, FALSE if not found

Output

```
[1] TRUE
[1] FALSE
```

ON-SITE QUESTIONS

ON-SITE QUESTION 1

Tutorial 1 On-site Questions

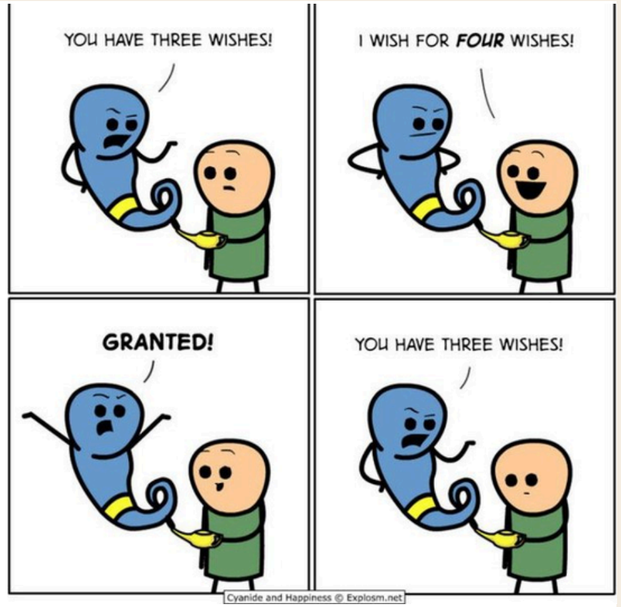
1. A sequence is generated using the following recursive relation

$$x_n = 2x_{n-1} - x_{n-2} + 5, \quad \text{for} \quad n \geq 3,$$

with $x_1 = 0$ and $x_2 = 1$.

(a) Use "for" loop in R to find the 30th term of the series.

(b) Find the smallest value of n such that $x_n \geq 1,000$.



Discuss with your group! Time limit: 12 mins

[ON-SITE 1A] BREAKDOWN

1. A sequence is generated using the following recursive relation

$$x_n = 2x_{n-1} - x_{n-2} + 5, \quad \text{for} \quad n \geq 3,$$

with $x_1 = 0$ and $x_2 = 1$.

(a) Use "for" loop in R to find the 30th term of the series.

1. I know $x_1 = 0$ and $x_2 = 1$

2. I want to find x_{30}

3. $x_{30} = 2x_{29} - x_{28} + 5$

4. I need to find $x_3, x_4 \ldots, x_{29}$ to obtain x_{30}

[ON-SITE 1A] BREAKDOWN

What do I need to do?

1. Define a vector to store $x_1, x_2, \ldots, x_{30}$
2. Assign $x_1 = 0$ and $x_2 = 1$ inside the vector
3. Loop through $x_n = 2x_{n-1} - x_{n-2} + 5$ where n is from 3 to 30,
   which helps us obtain $x_3, x_4 \ldots, x_{30}$

How do I convert this idea into R code?

[ON-SITE 1A] ANSWER

```r
# 1: Create vector to store x[1] to x[30]

x <- numeric(30)
```

numeric(30): create a vector with 30 elements, all zeros

Alternatives for numeric(30):

1. vector('numeric', 30)
2. rep(0,30)

[ON-SITE 1A] ANSWER

```r
# 1: Create vector to store x[1] to x[30]

x <- numeric(30)

# 2: Assign x1 = 0 and x2 = 1

x[1] <- 0
x[2] <- 1
```

[ON-SITE 1A] ANSWER

```r
# 1: Create vector to store x[1] to x[30]

x <- numeric(30)

# 2: Assign x1 = 0 and x2 = 1

x[1] <- 0
x[2] <- 1

# 3. Loop through x[n] <- 2*x[n-1] - x[n-2] + 5 where n is from 3 to 30

for (n in 3:30) {

  x[n] <- 2*x[n-1] - x[n-2] + 5

}
```

```r
for (variable in sequence) {

  # Code to execute

}
```

[ON-SITE 1A] ANSWER

```r
# 1: Create vector to store x[1] to x[30]

x <- numeric(30)

# 2: Assign x1 = 0 and x2 = 1

x[1] <- 0
x[2] <- 1

# 3. Loop through x[n] <- 2*x[n-1] - x[n-2] + 5 where n is from 3 to 30

for (n in 3:30) {

  x[n] <- 2*x[n-1] - x[n-2] + 5

}

# 4. Obtain x[30]

x[30] # 2059
```

Ans: 2059

[ON-SITE 1A] EXTRA THINKING

Is there a way to solve Q1A without needing to define a vector to store x[1] to x[30]? (aka a way without needing to call numeric(30))

Feel free to share your answer/ discuss with me after class!

[ON-SITE 1B] BREAKDOWN & ANSWER

(b) Find the smallest value of n such that $x_n \geq 1,000$.

1. The current x that we have:

```
> x
 [1]    0    1    7   18   34   55   81  112  148  189  235  286  342
[14]  403  469  540  616  697  783  874  970 1071 1177 1288 1404 1525
[27] 1651 1782 1918 2059
```

2. Whether each element in x fulfills x >= 1000

```
> x>=1000
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
[23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

[ON-SITE 1B] BREAKDOWN & ANSWER

(b) Find the smallest value of n such that $x_n \geq 1,000$.

3. Indices (Positions) of elements that fulfill x >= 1000

```
> which(x>=1000)
[1] 22 23 24 25 26 27 28 29 30
```

4. Get the smallest value of n that fulfills the condition using min()

```
> min(which(x >= 1000))
[1] 22
```

Ans: 22

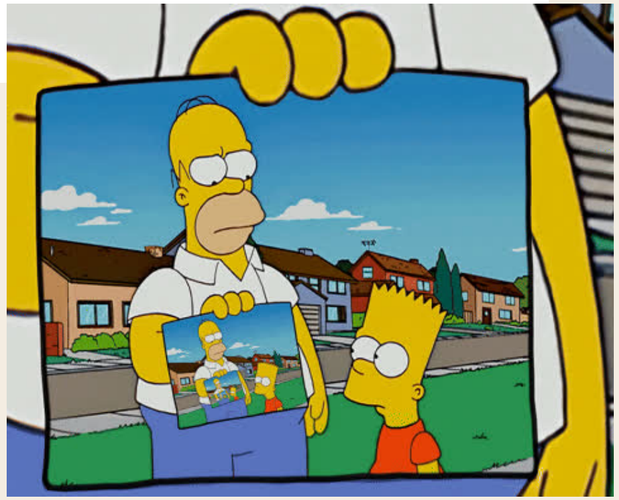
ON-SITE QUESTION 2

2. Consider another sequence which is generated using the following recursive relation

$y_1 = 2800 + 1.02 \times y_0$, with $y_0 = 10000$ and

$$y_n = 2800 + 1.02 \times y_{n-1}, \quad \text{for} \quad n \geq 2.$$

Find the smallest value of n such that $y_n \geq 300,000$.



Discuss with your group! Time limit: **7 mins (Challenge!)**

[ON-SITE 2] BREAKDOWN

This is similar to question 1!

But this time you don't know the **size of vector to define** before you start finding the smallest n (eg. vector size = 30 in q1).

You can either:

1. Try different vector sizes till you find one which last element is $\geqslant 300000$
2. Use a **while loop**!

[ON-SITE 2] BREAKDOWN

Method 1: Try different vector sizes until last element $\geqslant 300000$

```r
y <- numeric(50) # Try 50 first

# y[0] <- 10000 (unlike python, there's no 0 index in R)
y[1] <- 2800 + (1.02 * 10000)

for (n in 2:50) {
  y[n] <- 2800 + (1.02 * y[n-1])
}

y[50] # 263738.2 (does not exceed 300000 yet)
```

[ON-SITE 2] BREAKDOWN

Method 1: Try different n values until fulfills $y_n \geqslant 300000$

```r
y <- numeric(70) # Try 70 this time

y[1] <- 2800 + (1.02 * 10000)

for (n in 2:70) {
  y[n] <- 2800 + (1.02 * y[n-1])
}

y[70] # 459933.7, exceeded

min(which(y>=300000)) # 55
```

[ON-SITE 2] BREAKDOWN

Method 2: Use a while loop

```r
z <- 10000
i <- 0 # i helps us keep track of the indices

while(z < 300000) {
  z <- 2800 + (1.02 * z)
  i <- i + 1
}

i # 55
z # 305759.6
```

Ans: 55

- While z < 300000, loop the recursive formula until **z >= 300000**.
- Track the number of loops using **i**. When z >= 300000, i is the indices of the exceeded value

(which is also the minimum indices which fulfills z >= 300000)

[ON-SITE 2] BREAKDOWN

Method 2: Use a while loop

```
y <- numeric()
y <- append(y, 2800 + 1.02*10000)

while (max(y) < 300000) {
  y <- c(y, 2800 + 1.02*max(y))
}

length(y) # 55
y[55] # 305759.6
```

Ans: 55

Prof Daisy's official solution

OFF-SITE QUESTIONS

OFF-SITE QUESTION 1A

1. General idea of this problem: You have just graduated from NUS and just started your first job. You plan to buy a flat on your own which has **price** = $1,200,000 (1.2 million dollars). You need to save money for several years before you can afford to make the down payment which is 25% of the flat's price.

   - Call the amount that you have saved thus far: **saved**. You start the very first month with a savings of $10,000 that your parents gave you.
   - Call your monthly salary as **salary** which is paid at the end of every month.
   - Each month, you are going to dedicate 40% of your salary to save for the down payment.
   - Assume that you invest your savings wisely, with a monthly average return of 2%. That means: at the end of each month, you receive an additional of **saved** $\times 0.02$ funds where **saved** is the amount you have from end of previous month to put into your savings.
   - At the end of each month, your savings will be increased by the return on your investment, plus 40% of your monthly salary.

   Note: In your code for the questions below, you MUST use the names as given in bold above.

   (a) Write the code to calculate how many months it will take you to save up enough money for the down payment for two persons of of different salary: (i) **salary** = $7,000; and (ii)**salary** = $10,000.

OFF-SITE QUESTION 1A

General idea of this problem: You have just graduated from NUS and just started your first job. You plan to buy a flat on your own which has **price** = $1,200,000 (1.2 million dollars). You need to save money for several years before you can afford to make the down payment which is 25% of the flat's price.

- Call the amount that you have saved thus far: **saved**. You start the very first month with a savings of $10,000 that your parents gave you.
- Call your monthly salary as **salary** which is paid at the end of every month.
- Each month, you are going to dedicate 40% of your salary to save for the down payment.
- Assume that you invest your savings wisely, with a monthly average return of 2%. That means: at the end of each month, you receive an additional of **saved** ×0.02 funds where **saved** is the amount you have from end of previous month to put into your savings.
- At the end of each month, your savings will be increased by the return on your investment, plus 40% of your monthly salary.

Note: In your code for the questions below, you MUST use the names as given in bold above.

(a) Write the code to calculate how many months it will take you to save up enough money for the down payment for two persons of of different salary: (i) **salary** = $7,000; and (ii)**salary** = $10,000.

- down-payment = 25% of $1.2 million
- saved (first month) = $10,000
- salary = $7000, $10000
- saved(month+1) = saved(month) + ((saved(month) * 0.02) + (0.4 × salary))

How many months to fully pay down-payment?

OFF-SITE QUESTION 1A

- down-payment = 25% of $1.2 million
- saved (first month) = $10,000
- salary = $7000, $10000
- saved(month+1) = saved(month) + ((saved(month) x 0.02) + (0.4 × salary))

How many months to fully pay down-payment?

- down-payment = $300,000
- saved[0] = $10,000
- salary = $7000, $10000
- saved[n+1] = saved[n] + ((saved[n] × 0.02) + (0.4 × salary))

Smallest n that fulfills n >= 300,000

Question may look very long and complicated. But if we break it down, it's similar to what we worked on before!

[OFF-SITE Q1A] BREAKDOWN

Start by defining variable names given in the question

```
price <- 1200000
saved <- 10000
salary <- 7000
return <- 0.02

downpayment <- 0.25*price
savedsalary <- 0.4*salary
```

[OFF-SITE Q1A] ANSWER

Like the previous question, let's solve it using while loop!

```
month <- 0 # start at month 0

while (saved < downpayment) {

  saved <- saved + ((return * saved) + (savedsalary))

  month <- month + 1 # month increases by 1 each time we loop

}

month # 55 months
```

Ans: 55 months

(i) salary = 7000

[OFF-SITE Q1A] ANSWER

Like the previous question, let's solve it using while loop!

```r
month <- 0 # start at month 0

while (saved < downpayment) {

  saved <- saved + ((return * saved) + (savedsalary))

  month <- month + 1 # month increases by 1 each time we loop

}

month # 44 months
```

Ans: 44 months

(ii) salary = 10000

[OFF-SITE Q1A] EXTRA THINKING

In my code, I copied and pasted the exact same code lines for the second person (with salary = $10,000). What if I have 10 people, do I just copy and paste the code 10 times?

You obviously can do it, but code will be LONG. What can I do to resolve that?

[OFF-SITE Q1B] BREAKDOWN

(b) In question above, we unrealistically assumed that the salary doesn't change over the years. However, now we consider that the salary will be raised every 4 months by a rate named **rate**, this variable should be in decimal form (i.e. 0.03 for 3%). The new salary will be applied for the month after every batch of 4 months.

With this further assumption, write the code to calculate how many months it will take a person to save up enough money for the down payment if that person has (i) (**salary** = \$7,000 and **rate** = 0.02); (ii) (**salary** = \$10,000 and **rate** = 0.01).

WHAT CHANGED?

1. $saved = 10000$
2. $saved = saved + (return \times saved + savedsalary)$
3. Every 4 months: $salary = salary + (salary \times rate)$
4. Smallest month such that $saved \geqslant downpayment$

[OFF-SITE Q1B] BREAKDOWN

HOW TO CODE OUT ‘EVERY 4 MONTHS’?

```
if (month%%4 == 0)
```

%%: modulo (aka remainder of division)

[OFF-SITE Q1B] ANSWER

Step 1: Define all variables

```
price <- 1200000
saved <- 10000
salary <- 7000
return <- 0.02
rate <- 0.02

downpayment <- 0.25*price
savedsalary <- 0.4*salary
```

salary = $7000, rate = 0.02

Step 2: While loop to find months where they finally earned enough to pay down-payment

```
month <- 0 # start at month 0

while (saved < downpayment) {

  saved <- saved + ((return * saved) + (savedsalary))

  month <- month + 1 # month increases by 1 each time we loop

  # increase salary every 4 months
  if (month %% 4 == 0) {
    salary <- salary + (salary * rate)
    savedsalary <- 0.4*salary # update savedsalary
  }
}

month # 52 months
```

Ans: 52 months

[OFF-SITE Q1B] ANSWER

Step 1: Define all variables

```r
price <- 1200000
saved <- 10000
salary <- 10000
return <- 0.02
rate <- 0.01

downpayment <- 0.25*price
savedsalary <- 0.4*salary
```

salary = $10000, rate = 0.01

Step 2: While loop to find months where they finally earned enough to pay down-payment

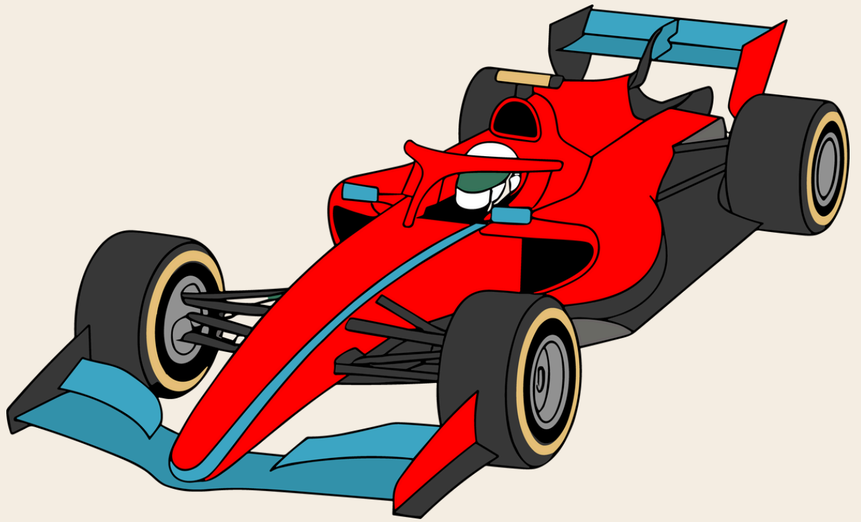
```r
month <- 0 # start at month 0

while (saved < downpayment) {

  saved <- saved + ((return * saved) + (savedsalary))

  month <- month + 1 # month increases by 1 each time we loop

  # increase salary every 4 months
  if (month %% 4 == 0) {
    salary <- salary + (salary * rate)
    savedsalary <- 0.4*salary # update savedsalary
  }
}

month # 43 months
```

Ans: 43 months

OFF-SITE QUESTION 2

For Question 1(a) above, use the code to define a function, called F1, where the argument of F1 is **salary**; the output of function F1 is the number of months. Run function F1 for the two cases mentioned to obtain the results.

- Function name: F1
- Argument/Input: salary
- Output: number of months



Format of function in R:

```
func_name <- function(input) {
    ....
    return(output)
}
```

[OFF-SITE Q2] ANSWER

```r
# Step 1: Define variables (except salary)
price <- 1200000
downpayment <- 0.25*price

# Step 2: Create function F1
F1 <- function(salary) {

  saved <- 10000 # saved inside function cause it always starts at $10000
  savedsalary <- 0.4*salary
  month <- 0

  while (saved < downpayment) {
    saved <- saved + ((return * saved) + (savedsalary))
    month <- month + 1 # month increases by 1 each time we loop
  }

  return(month)
}

F1(7000) # 55
F1(10000) # 44
```

Q: IS THERE A WAY TO REMOVE STEP 1?

[OFF-SITE Q2] ANSWER

```r
F1 <- function(salary, price = 1200000) {

  downpayment <- 0.25*price
  saved <- 10000
  savedsalary <- 0.4*salary
  month <- 0

  while (saved < downpayment) {
    saved <- saved + ((return * saved) + (savedsalary))
    month <- month + 1 # month increases by 1 each time we loop
  }

  return(month)
}

F1(7000) # 55
F1(10000) # 44
```

Ans: 55, 44

** price will be 1200000 if we don't define it when we call the F1 function. But when we do, it changes to the value we defined.

Try: F1(7000, price=2500000)

ANS: DEFINE IT INTO FUNCTION AS OPTIONAL ARGUMENTS

OFF-SITE QUESTION 3

For Question 1(b) above, use the code to define a function, called F2, where F2 has two arguments: **salary** and **rate** where the default value for **rate** is 0.02; the output of function F2 is the number of months. Run function F2 for the two cases mentioned to obtain the results.

- Function name: F2
- Argument/Input: salary, rate (default = 0.02)
- Output: number of months

OFF-SITE QUESTION 3

```r
F2 <- function(salary, price = 1200000, rate = 0.02) {

  downpayment <- 0.25*price
  saved <- 10000
  savedsalary <- 0.4*salary
  month <- 0

  while (saved < downpayment) {
    saved <- saved + ((return * saved) + (savedsalary))
    month <- month + 1 # month increases by 1 each time we loop

    if (month %% 4 == 0) {
      salary <- salary + (salary * rate)
      savedsalary <- 0.4*salary # update savedsalary
    }
  }

  return(month)
}

F2(salary = 7000, rate = 0.02) # answer: 52
F2(salary = 10000, rate = 0.01) # answer: 43
```

Ans: 52, 43

EXTRA QUESTIONS

EXTRA QUESTION

Example Code

```
vec <- c(1,2,3,4)

for (element in vec) {

  vec <- vec[-element]

}

print(vec)
```

Output

```
[1] 2 4
```

FAQ

Can I use = and <- interchangeably during assignments of variables?

Yes!

Example Code

```
x = 6
y <- 7
```

Under Environment

Values	
x	6
y	7