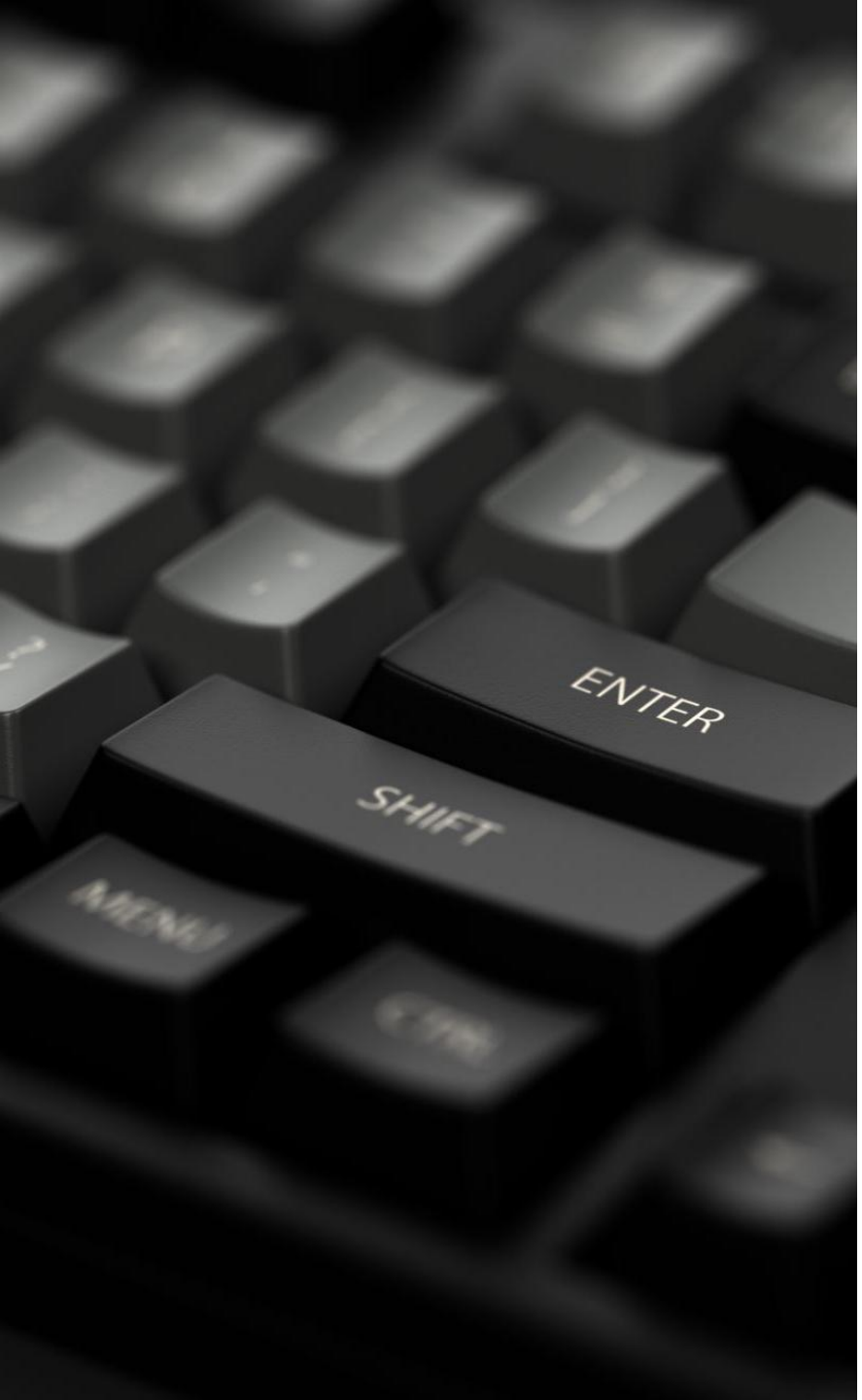
A gamified introduction to Python Programming

Lecture 3
None, pass, Booleans and functions

Matthieu DE MARI – Singapore University of Technology and Design



Outline (Chapter 3)

- What are the **None** type and the **pass** keyword?
- What is the **Boolean** type?
- **Boolean quiz** for practice.
- **Advanced concepts on Booleans**.
- What are **functions** in Python and how to use them?
- What are **memory states in functions**?

The None type

Definition (the None type):
You can define a variable with an empty value by assigning a **None** value to a variable. This is used to **indicate that a variable** exists in memory even though **no value has yet been assigned** to it.

Used in some activities to “hint” on some variables, functions or structures you should be using but should probably be replaced with something else!

```
# Creating a None variable, with None as value and a None type
variable = None
print(variable)
print(type(variable))
```

```
None
<class 'NoneType'>
```

The pass keyword

Definition (the pass keyword): Similarly, you can use the **pass** keyword, which is equivalent to "doing nothing". It simply has no effect on the code.

Used in some activities to "hint" on some variables, functions or structures you should be using but should probably be replaced with something else!

```
# First print
print("Hello")
# Has no effect
pass
# Second print
print("Hello again")
```

```
Hello
Hello again
```

The

Defin
Simil
keyw
"doin
effec

Used
some
struc
shou
some

```python
# 1. Let us choose a number to guess (for instance, 6)
# Feel free to change it to a different value if you want!
true_number = 6

# 2. Ask for user to guess the number, acquire it by using input()
# Remember to convert the string variable, you obtained from input(),
# to an int variable before performing boolean operations!
# Your turn to code below!
guessed_number = None

# 3. Check if the user guessed the right number
# Your turn to code below!
print(???)

# 4. Check if the guessed_number is striclty lower than the true_number
# Your turn to code below!
print(None)

# 5. Check if the guessed_number is striclty higher than the true_number
# Your turn to code below!
pass
```

```
in")
```

The boolean type

Definition (the boolean type):
The **boolean** (bool) type is an essential type in programming.

In simple terms, it is the answer to a yes or no question.

It is commonly used **to check if a condition is satisfied** or not.

It can only take two values:

- **True,** if the condition is satisfied;
- or **False,** otherwise.

```
bool1 = True
bool2 = False
print(bool1)
print(bool2)
print(type(bool1))
print(type(bool2))
```

```
True
False
<class 'bool'>
<class 'bool'>
```

Boolean comparisons

Definition (the == operator):

The == operator is used to check if the two variables, on both sides of the == sign, have identical values.

The result of this operation is a Boolean variable, with value:

- **True**, if both variables have **identical values**;
- and **False**, **otherwise**.

```
a = 1
b = 1
c = 2
bool1 = (a == b)
bool2 = (a == c)
print(bool1)
print(bool2)
```

```
True
False
```

Boolean comparisons

Important note: Almost always, it returns False, by default, if both variables have different types.

One of the few exceptions to this rule is that it works if variables have mixed int/float types.

In that case, the numerical value is being compared.

```
a = 1.0
b = 1
c = 2
bool1 = (a == b)
bool2 = (a == c)
print(type(a))
print(type(b))
print(type(c))
print(bool1)
print(bool2)
```

```
<class 'float'>
<class 'int'>
<class 'int'>
True
False
```

Boolean comparisons

Important note: Almost always, it returns False, by default, if both variables have different types.

One of the few exceptions to this rule is that it works if variables have mixed int/float types.

In that case, the numerical value is being compared.

But **not with str and int/float!**

```
a = 1.0
b = 1
c = 2
bool1 = (a == b)
bool2 = (a == c)
print(type(a))
print(type(b))
print(type(c))
print(bool1)
print(bool2)
```

```
<class 'float'>
<class 'int'>
<class 'int'>
True
False
```

```
a = "1"
b = 1.0
c = 1
bool1 = (a == b)
bool2 = (a == c)
print(type(a))
print(type(b))
print(type(c))
print(bool1)
print(bool2)
```

```
<class 'str'>
<class 'float'>
<class 'int'>
False
False
```

Boolean comparisons

Following the same logic as the **==** operator, we can define, the following operators, **for numerical types (int/float).**

- **!=:** to check if two variables are **different**.

Note: Again, careful with the types on both sides! Let us only use these operators with similar types on both sides!

```
a = 1
b = 1
c = 2
bool1 = (a != b)
bool2 = (a != c)
print(bool1)
print(bool2)
```

```
False
True
```

```
a = "1"
b = 1
c = 1.0
bool1 = (a != b)
bool2 = (a != c)
bool3 = (b != c)
print(bool1)
print(bool2)
print(bool3)
```

```
True
True
False
```

Boolean comparisons

Similarly,

- **>:** if variable on the left-hand side has a **strictly higher numerical value**, than the one on the right-hand side.
- **<:** same as **>**, but checking for **strictly lower numerical value**.

Note: again, careful with the types on both sides! Let us only use these operators with similar types on both sides!

```
a = "1"
b = 1
print(type(a))
print(type(b))
bool1 = (a > b)
print(bool1)
```

```
<class 'str'>
<class 'int'>

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-dedd4801f811> in <module>
      3 print(type(a))
      4 print(type(b))
----> 5 bool1 = (a > b)
      6 print(bool1)

TypeError: '>' not supported between instances of 'str' and 'int'
```

Boolean comparisons

Similarly,

- **>=:** if variable on the left-hand side has a **higher or equal numerical value**, than the one on the right-hand side.
- **<=:** same as **>=**, but checking for **lower or equal numerical value**.

Note: again, careful with the types on both sides! Let us only use these operators with similar types on both sides!

```
a = 1
b = 1
c = 2
bool1 = (a >= b)
bool2 = (a >= c)
print(bool1)
print(bool2)
```

```
True
False
```

```
a = 1
b = 1
c = 2
bool1 = (a <= b)
bool2 = (a <= c)
print(bool1)
print(bool2)
```

```
True
True
```

Boolean combinations: and, or, not

Definition (the and operator):

The **and** operator returns a boolean with value **True**, if and only if **both Boolean variables on the left- and right-hand sides of the and operator are True**.

It returns **False** otherwise.

Typically used to ask more advanced questions (e.g. voting rights if this person is at least 18yo and a Singaporean citizen?)

```
bool1 = True
bool2 = True
print(bool1 and bool2)

bool1 = True
bool2 = False
print(bool1 and bool2)

bool1 = False
bool2 = True
print(bool1 and bool2)

bool1 = False
bool2 = False
print(bool1 and bool2)
```

```
True
False
False
False
```

Boolean combinations: and, or, not

Definition (the or operator):

Similarly, the **or** operator returns a boolean with value **True**, if and only if **at least one of the Boolean variables on the left- and right-hand sides of the or operator are True**.

It returns **False** otherwise.

Again, typically used to ask more advanced questions.

```
bool1 = True
bool2 = True
print(bool1 or bool2)

bool1 = True
bool2 = False
print(bool1 or bool2)

bool1 = False
bool2 = True
print(bool1 or bool2)

bool1 = False
bool2 = False
print(bool1 or bool2)
```

```
True
True
True
False
```

Boolean combinations: and, or, not

Definition (the or operator):

Similarly, the **or** operator returns a boolean with value **True**, if and only if **at least one of the Boolean variables on the left- and right-hand sides of the or operator are True**.

It returns **False** otherwise.

Again, typically used to ask more advanced questions.

Boolean combinations: and, or, not

Definition (the not operator):

The **not** operator returns a boolean with **opposite value.**

- If **var** is **True,** then **not var** is **False;**
- And, if **var** is **False,** then **not var** is **True.**

```
bool1 = True
bool2 = False
print(not bool1)
print(not bool2)
```

```
False
True
```

slido

Please download and install the Slido app on all computers you use

Assume that a, b and c are three int variables with respective values 6, 3 and 9. What will be displayed upon running the command print(a+b==c)?

ⓘ Start presenting to display the poll results on this slide.

Boolean type: practice quiz! (answers)

```
a = 6
b = 3
c = 9
# Q1: is the result of (a + b) equal to c?
bool1 = (a + b == c)
print(bool1)
```

```
True
```

slido

Please download and install the Slido app on all computers you use

Assume that a is an int variable with value 6. What will be displayed upon running the command print(a%2==0)?

ⓘ Start presenting to display the poll results on this slide.

Boolean type: practice quiz! (answers)

```
a = 6

bool1 = (a % 2 == 0)
print(bool1)
```

```
True
```

slido

Please download and install the Slido app on all computers you use

Which int values of the variable a will display True when running the command print(a%2==0)?

ⓘ Start presenting to display the poll results on this slide.

Boolean type: practice quiz! (answers)

```
a = 6
# Q2: is a an even number?
bool1 = (a % 2 == 0)
print(bool1)
```

```
True
```

Boolean type: practice quiz! (answers)

```
a = 6
b = 7
# Checking for even numbers in Python
bool1 = (a % 2 == 0)
bool2 = (b % 2 == 0)
print(bool1)
print(bool2)
# Checking for odd numbers in Python
bool3 = (a % 2 == 1)
bool4 = (b % 2 == 1)
print(bool3)
print(bool4)
```

```
True
False
False
True
```

slido

Please download and install the Slido app on all computers you use

Assume that a, b and c are three int variables with respective values 1, 2 and 3. What will be displayed upon running the command print((a>b) and (c>b))?

ⓘ Start presenting to display the poll results on this slide.

Boolean type: practice quiz! (answers)

```
a = 1
b = 2
c = 3
# Q3: are both a and c strictly greater than b?
bool1 = ((a>b) and (c>b))
print(bool1)
print(a>b)
print(c>b)
```

```
False
False
True
```

slido

Please download and install the Slido app on all computers you use

Assume that a, b, c and d are four int variables with respective values 1, 2, 3 and 4. What will be displayed upon running the (absolutely awful) command, below?

print((not (a>=b) or (c<b)) and (d+3>=c*2))

ⓘ Start presenting to display the poll results on this slide.

Boolean type: practice quiz! (answers)

```
a = 1
b = 2
c = 3
d = 4
# Q4: Too many things going on at the same time!
# (Break it down into substeps!)
bool1 = (((not a>=b) or (c<b)) and (d+3 >= c*2))
print(bool1)
```

```
True
```

Boolean type: practice quiz! (answers)

```
a = 1
b = 2
c = 3
d = 4
# Q4: Too many things going on at the same time!
# (Break it down into substeps!)
bool1 = (((not a>=b) or (c<b)) and (d+3 >= c*2))
print(bool1)
```

```
True
```

```
# Q4: breaking it down
bool2 = (not a>=b)
print(bool2)
bool3 = (c<b)
print(bool3)
bool4 = bool2 or bool3
print(bool4)
bool5 = d+3 >= c*2
print(bool5)
bool6 = bool4 and bool5
print(bool6)
```

```
True
False
True
True
True
```

Boolean conversion

Bool → Int/Float: You can convert a Boolean into an int/float number.

- **True** transforms into **1 (int)** or **1.0 (float).**
- **False** transforms into **0 (int)** or **0.0 (float).**

```
bool1 = True
bool2 = False
int1 = int(bool1)
int2 = int(bool2)
print(int1)
print(int2)
```

```
1
0
```

```
bool1 = True
bool2 = False
float1 = float(bool1)
float2 = float(bool2)
print(float1)
print(float2)
```

```
1.0
0.0
```

Boolean conversion

Bool → Int/Float: You can convert a Boolean into an int/float number.

- **True** transforms into **1 (int)** or **1.0 (float).**
- **False** transforms into **0 (int)** or **0.0 (float).**

Fun fact: that is reason behind the on/off symbols.

On/True/1
Off/False/0

Boolean conversion

Int/Float → Bool: You can convert an int/float number into a boolean.

- Any non-zero numerical value becomes **True.**
- A zero numerical value becomes **False.**

```
a = 1
b = 0
c = 0.1154654
d = 0.0
print(bool(a))
print(bool(b))
print(bool(c))
print(bool(d))
```

```
True
False
True
False
```

slido

Please download and install the Slido app on all computers you use

Assume that a, b and c are three int variables with values 1, 2 and 3 respectively. What will be displayed upon running the command print(a>b and c)?

ⓘ Start presenting to display the poll results on this slide.

slido

Please download and install the Slido app on all computers you use

Assume that a, b and c are three int variables with values 1, 2 and 3 respectively. What will be displayed upon running the command print(a<b and c)?

Note that the comparison sign has been changed compared to the previous question.

ⓘ Start presenting to display the poll results on this slide.

Short-circuit evaluation

Definition (Short-circuit evaluation):

Short-circuit evaluation in Python means that in boolean expressions using the "**and**" keyword and "**or**" keyword, the evaluation might stop prematurely.

- For example, in the expression "**a and b**", if **a** is **False**, **b** is not evaluated because the entire expression is guaranteed to be **False**.
- Similarly, in "**a or b**", if **a** is **True**, **b** is not evaluated because the entire expression is guaranteed to be **True**.

Short-circuit evaluation

Definition (Short-circuit evaluation):

Short-circuit evaluation in Python means that in boolean expressions using the "**and**" keyword and "**or**" keyword, the evaluation might stop prematurely.

- For example, in the expression "**a and b**", if **a** is **False**, **b** is not evaluated because the entire expression is guaranteed to be **False**.
- Similarly, in "**a or b**", if **a** is **True**, **b** is not evaluated because the entire expression is guaranteed to be **True**.
- **Typical mistake:** HOWEVER, in the expression "**a and b**", if **a** is **True**, then the result of **"a and b**" will be exactly equal to whatever **b** is (True or False). Funny enough, if b is not a Boolean (but say an int with value 4 instead), then the whole expression "**a and b**" will be evaluated as 4 (wut !?!?).

Activity 1: Guess the number game!

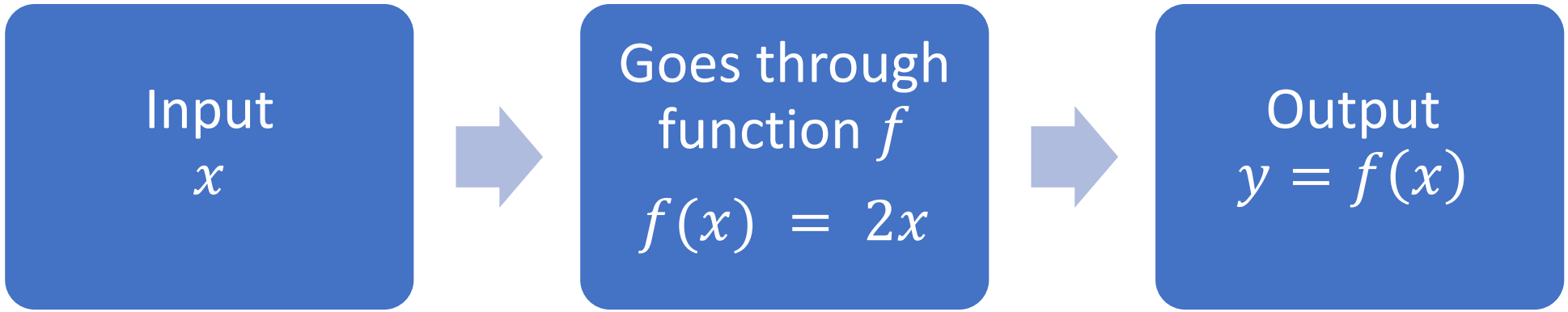
Let us play a bit with the concepts, with a first activity

You can find it in the notebook
Activity 1 - Guess the number game.ipynb

Function objects

In mathematics, we often like to define **functions**, for instance:

$$y = f(x) = 2x$$

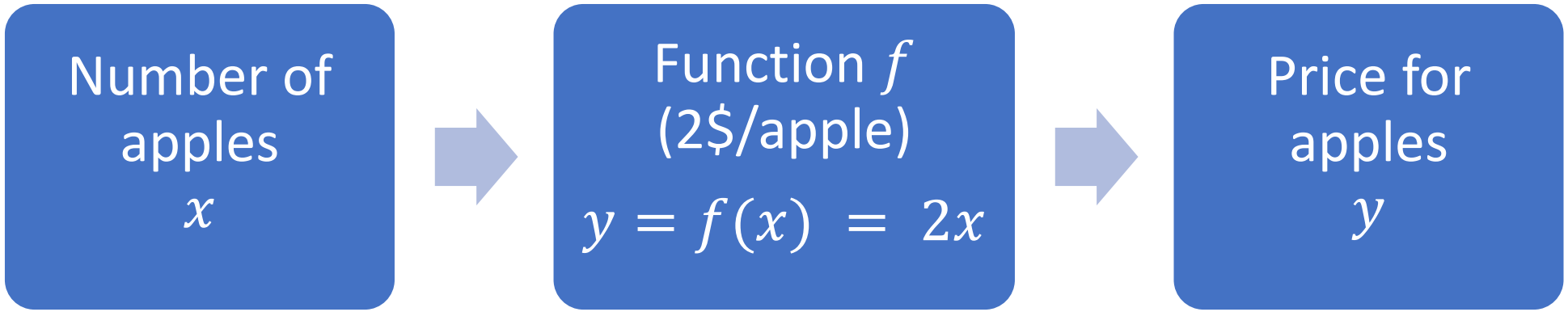


The same can be done in Python, by creating a function.

Function objects

In mathematics, we often like to define **functions**, for instance:

$$y = f(x) = 2x$$



For instance: Let us say we want to write a function to calculate the price of a bag of apples (2$ per apple in the bag).

Note: Most concepts in real life can be described as functions.

Function formatting in Python

Definition (Python functions):

A Python **function** is a block of reusable code which only runs when it is **called**.

You can **pass data**, known as **parameters, arguments** or **input values**, into a function (e.g., the value *number_apples* here).

A function can **return data**, as a **result** or **output value** (e.g., the value *price* here).

```
def price_apples(number_apples):
    price = 2*number_apples
    return price
```

```
number1 = 2
price1 = price_apples(number_apples = number1)
print(price1)
```

```
4
```

Function formatting in Python

Writing a function 101:

- You can define a function, with a **def** statement.
- After **def**, type the **function's name** (appears in blue).
- **Between parentheses**, after the function's name, type **arguments/parameters**.

```
def price_apples(number_apples):
    price = 2*number_apples
    return price
```

```
number1 = 2
price1 = price_apples(number_apples = number1)
print(price1)
```

```
4
```

Note: Arguments/parameters would be values that the function needs to operate. In our example, it is not possible to compute the price for a bag of apples without knowing the number of apples in the bag.

Function formatting in Python

Writing a function 101:

- **Using the return keyword**, type **output values/results,** that your function should give, if any.

```
def price_apples(number_apples):
    price = 2*number_apples
    return price
```

```
number1 = 2
price1 = price_apples(number_apples = number1)
print(price1)
```

```
4
```

Note: In general, the returned values are values of interest that we are aiming to compute with the given function. They are also values that could later be reused by some other functions. For instance, we could have a second function for printing a receipt, and it would need to know the price we just calculated.

Function formatting in Python

Note 1: Your functions may have multiple inputs, as well as outputs, separated with commas.

Note 2: When calling a function, you can

- Explicitly assign parameter, e.g. f2(val1=x1, val2=x2),
- Rely on positional arguments, e.g. f2(x1, x2), where Python automatically assigns values in the order that they appear in the **def** statement.

```
def f2(val1, val2):
    sum_val = val1 + val2
    mult_val = val1*val2
    return sum_val, mult_val
```

```
x1 = 2
x2 = 3
y1, y2 = f2(val1 = x1, val2 = x2)
print(y1)
print(y2)
```

```
5
6
```

```
x1 = 2
x2 = 3
y1, y2 = f2(x1, x2)
print(y1)
print(y2)
```

```
5
6
```

Function formatting in Python

Note 3: Your function may also not **return** anything (that is the case for the **print()** function!).
It is equivalent to **return None**.

```
output = print("Hello")
```

```
Hello
```

```
print(output)
```

```
None
```

```
def say_hello():
    print("Hello!")
```

```
output = say_hello()
```

```
Hello!
```

```
print(output)
```

```
None
```

Function formatting in Python

Note 3: Your function may also not **return** anything (that is the case for the **print()** function!).
It is equivalent to **return None**.

```
output = print("Hello")
```

```
Hello
```

```
print(output)
```

```
None
```

```
def say_hello():
    print("Hello!")
```

```
output = say_hello()
```

```
Hello!
```

```
print(output)
```

```
None
```

Note 4: In-between the def and return statement (if any), you may write lines of code to perform any number of additional or intermediate tasks.

Important, however, lines of code inside the function are **indented** with 4 spaces.

```
def f4(x):
    print("Can you see this?")
    return 2*x
    # The print below will never be executed
    print("How about this?")
```

```
x = 2
y = f4(x)
```

```
Can you see this?
```

Returning vs. Printing

Note 5: Returning and printing are not the same thing!

- **Printing** simply means “displaying the current value of something on screen”.
- **Returning** means producing a specific value, deemed the final result of the function.
- The value being returned can then be caught and stored in memory, which would not be the case if the return had been omitted.
- These values might often by used by additional functions (e.g. price of apples, printing a receipt, etc.).

```
def f5(val):
    print(val)
    return val*2
```

```
x1 = 2
y1 = f5(x1)
print(y1)
```

```
2
4
```

Function formatting in Python

Note 6: Once a **return** is reached and executed, the function closes. It will not execute anything else.

Any operations that happen after the **return** keyword triggers will simply not execute. Here, the second print does not show because the **return** stopped the function.

```
def f4(x):
    print("Can you see this?")
    return 2*x
    # The print below will never be executed
    print("How about this?")
```

```
x = 2
y = f4(x)
```

```
Can you see this?
```

Variables in functions

Note 7: Variables defined **inside** the function are stored in memory while the function runs but are cleared/discarded once the end of the function is reached.

If you need to access a variable defined inside a function, after the function has executed, then it needs to be explicitly returned.

```
def f5(val):
    print(val)
    return val*2
```

```
x1 = 2
y1 = f5(x1)
```

```
2
```

```
print(val)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-11-54a7d177c749> in <module>
----> 1 print(val)

NameError: name 'val' is not defined
```

Variables in functions and memory state

Computer memory state

Python script

```
1 def my_function(x):
2       y = 2*x
3       z = y + 3
4       return z
5
6 x1 = 10
7 z1 = my_function(x1)
8 print(z1)
```

Variables in functions and memory state

Computer memory state

Global variables

my_function = Function of some sort defined on line 1.

Python script

```
1 def my_function(x):
2       y = 2*x
3       z = y + 3
4       return z
5
6 x1 = 10
7 z1 = my_function(x1)
8 print(z1)
```

Variables in functions and memory state

Computer memory state

Global variables

my_function = Function of some sort defined on line 1.

x1 = integer variable with value 10

Python script

```
1 def my_function(x):
2       y = 2*x
3       z = y + 3
4       return z
5
6 x1 = 10
7 z1 = my_function(x1)
8 print(z1)
```

IGNORED FOR NOW

Variables in functions and memory state

Computer memory state

Global variables

my_function = Function of some sort defined on line 1.

x1 = integer variable with value 10

Function variables (for function called on line 7)

Python script

```
1 def my_function(x):
2       y = 2*x
3       z = y + 3
4       return z
5
6 x1 = 10
7 z1 = my_function(x1)
8 print(z1)
```

Variables in functions and memory state

Computer memory state

Global variables

my_function = Function of some sort defined on line 1.

x1 = integer variable with value 10

Function variables (for function called on line 7)

x = integer variable with value 10

Python script

```
1 def my_function(x):
2       y = 2*x
3       z = y + 3
4       return z
5
6 x1 = 10
7 z1 = my_function(x1)
8 print(z1)
```

Variables in functions and memory state

Computer memory state

Global variables

my_function = Function of some sort defined on line 1.

x1 = integer variable with value 10

Function variables (for function called on line 7)

x = integer variable with value 10

y = integer variable with value 20

Python script

```
1 def my_function(x):
2       y = 2*x
3       z = y + 3
4       return z
5
6 x1 = 10
7 z1 = my_function(x1)
8 print(z1)
```

Variables in functions

Computer memory state

Global variables

my_function = Function of some sort defined on line 1.

x1 = integer variable with value 10

Function variables (for function called on line 7)

x = integer variable with value 10

y = integer variable with value 20

z = integer variable with value 23

Python script

```
1 def my_function(x):
2       y = 2*x
3       z = y + 3
4       return z
5
6 x1 = 10
7 z1 = my_function(x1)
8 print(z1)
```

Variables in functions and memory state

Computer memory state

Global variables

my_function = Function of some sort defined on line 1.

x1 = integer variable with value 10

z1 = integer with value 23

Function variables (for function called on line 7)

x = integer variable with value 10

y = integer variable with value 20

z = integer variable with value 23

Python script

```
1 def my_function(x):
2       y = 2*x
3       z = y + 3
4       return z
5
6 x1 = 10
7 z1 = my_function(x1)
8 print(z1)
```

Variables in functions and memory state

Computer memory state

Global variables

my_function = Function of some sort defined on line 1

Before ending the function, Python will clear all the variables related to this function to save space.

~~**Function variables (for function called on line 7)**~~

~~x = integer variable with value 10~~

~~y = integer variable with value 20~~

~~z = integer variable with value 23~~

Python script

```
1 def my_function(x):
2       y = 2*x
3       z = y + 3
4       return z
5
6 x1 = 10
7 z1 = my_function(x1)
8 print(z1)
```

Variables in functions and memory state

Computer memory state

Global variables

my_function = Function of some sort defined on line 1.

x1 = integer variable with value 10

z1 = integer with value 23

Python script

```
1 def my_function(x):
2       y = 2*x
3       z = y + 3
4       return z
5
6 x1 = 10
7 z1 = my_function(x1)
8 print(z1)
```

Variables in functions and memory state

Computer memory state

Global variables

my_function = Function of some sort defined on line 1.

x1 = integer variable with value 10

z1 = integer with value 23

Python script

```
1 def my_function(x):
2       y = 2*x
3       z = y + 3
4       return z
5
6 x1 = 10
7 z1 = my_function(x1)
8 print(z1)
```

Variables in functions and memory state

Computer memory state

Global variables

my_function = Function of some sort defined on line 1.

x1 = integer variable with value 10

z1 = integer with value 23

Python script

```
1 def my_function(x):
2       y = 2*x
3       z = y + 3
4       return z
5
6 x1 = 10
7 z1 = my_function(x1)
8 print(z1)
9 print(y) # No longer exists! Error!
```

Variables in functions and memory state

In other words…

What happens in ~~Vegas~~ the function,

Stays in ~~Vegas~~ the function.

(unless it is explicitly returned)

A note on global variables

Definition (global variables):

In Python, variables defined outside of any function are usually called **global variables**. They can be used anywhere, even in functions, BUT it is better practice to pass them as parameters to the function explicitly.

```
# Here, a is a global variable
a = 10
def f6(x):
    # Function f6 requires a, but it was not listed as inputs required by the function
    # Python will save the day by fetching the global variable a
    # (Serious malpractice, will not work in many other languages!)
    return a*x
print(f6(5))
```

```
50
```

A note on global variables

Important note: Relying on global variables is considered malpractice!

- **If a function needs a variable, it should be listed as inputs.**
- Many other programming languages (e.g. C) will not allow for variables to be used in functions without being passed as inputs explicitly!

```
# Here, a is a global variable
a = 10
def f6(x):
    # Function f6 requires a, but it was not listed as inputs required by the function
    # Python will save the day by fetching the global variable a
    # (Serious malpractice, will not work in many other languages!)
    return a*x
print(f6(5))
```

```
50
```

Matt’s Great advice

Matt’s Great Advice: Use functions to avoid repetitions of code

Find yourself **copy-pasting blocks of code, and only changing a few values in this block of code**?

Then, you probably need a **function** of some sort, which is called multiple times, but with different parameters.

Functions makes your coding **modular**, **easier**, and your code will look a lot more professional.

Matt's Great advice: example

```
student_A_math_grade = 85
student_A_physics_grade = 70
student_A_chemistry_grade = 75
student_A_sum_grade = student_A_math_grade + student_A_physics_grade + student_A_chemistry_grade
student_A_mean_grade = student_A_sum_grade/3
```

```
student_B_math_grade = 95
student_B_physics_grade = 65
student_B_chemistry_grade = 50
student_B_sum_grade = student_B_math_grade + student_B_physics_grade + student_B_chemistry_grade
student_B_mean_grade = student_B_sum_grade/3
```

```
def avg_grade(math_grade, phy_grade, chem_grade):
    return (math_grade + phy_grade + chem_grade)/3
```

```
student_A_mean_grade = avg_grade(student_A_math_grade, student_A_physics_grade, student_A_chemistry_grade)
student_B_mean_grade = avg_grade(student_B_math_grade, student_B_physics_grade, student_B_chemistry_grade)
```

Subfunctions and calling them in other functions

Definition (Function modularity): You may find useful to **define several functions and call them inside other functions**. Again, this makes the code more modular, professional and readable.

```
def sum_all_grades(math_grade, phy_grade, chem_grade):
    summed_grades = math_grade + phy_grade + chem_grade
    return summed_grades
def divide_grades(summed_grades, number_grades):
    avg_grade = summed_grades/number_grades
    return avg_grade
def avg_grade_v2(math_grade, phy_grade, chem_grade):
    summed_grades = sum_all_grades(math_grade, phy_grade, chem_grade)
    number_grades = 3
    avg_grade = divide_grades(summed_grades, number_grades)
    return avg_grade
```

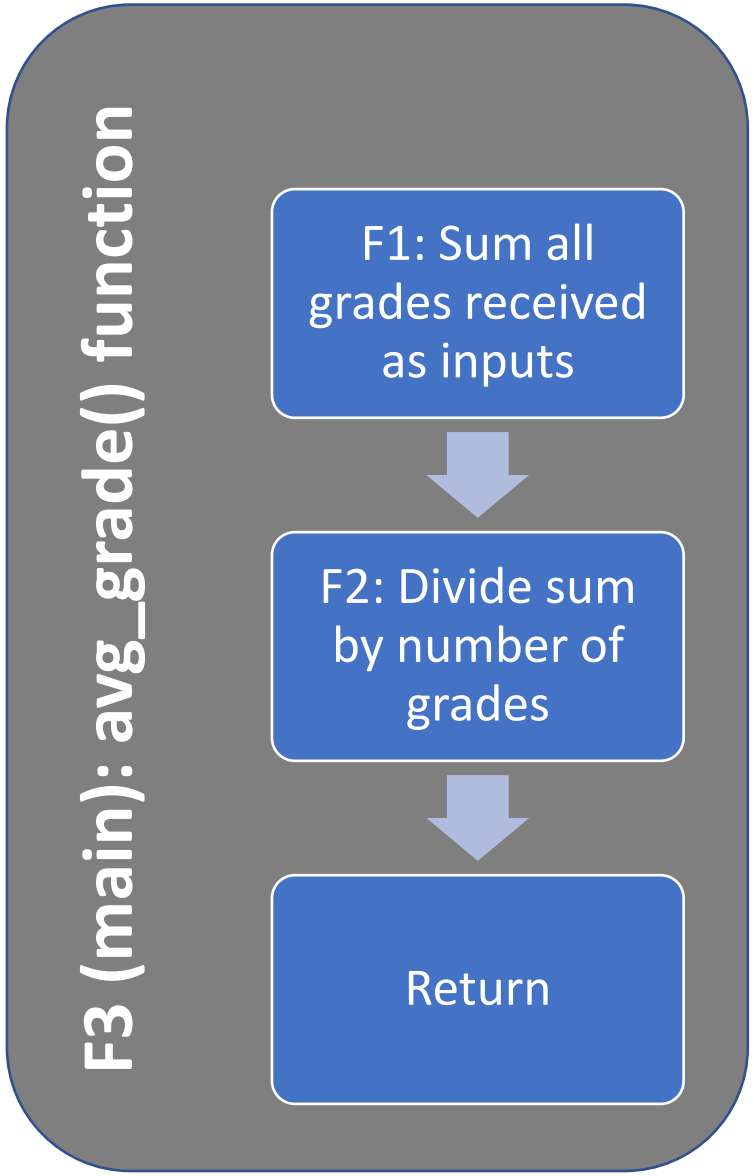
```
student_A_mean_grade = avg_grade_v2(student_A_math_grade, student_A_physics_grade, student_A_chemistry_grade)
student_B_mean_grade = avg_grade_v2(student_B_math_grade, student_B_physics_grade, student_B_chemistry_grade)
print(student_A_mean_grade)
print(student_B_mean_grade)
```

```
76.66666666666667
70.0
```

Functional diagrams

If you have to design a function and/or several subfunctions, you might find it useful to draw a **functional diagram**.

- See the one on the right, for the avg_grade() functions and its subfunctions.
- A bit overkill right now, but….
- When your code becomes heavier, breaking it down into well-chosen subfunctions, and keeping track of these functions will become essential.



Matt’s Great advice

Matt’s Great Advice: one main function to rule them all!

You may have defined several **subfunctions** and have designed a nice **modular** code.

Now is the time to define a **main function** that runs all of these functions at once. We often like to call it **main()**.

Activity 2: Ballistics of an angry bird

Let us practice these concepts with a second activity.

Check the notebook

Activity 2 - Ballistics of an angry bird.ipynb

Activity 2: Ballistics of an angry bird

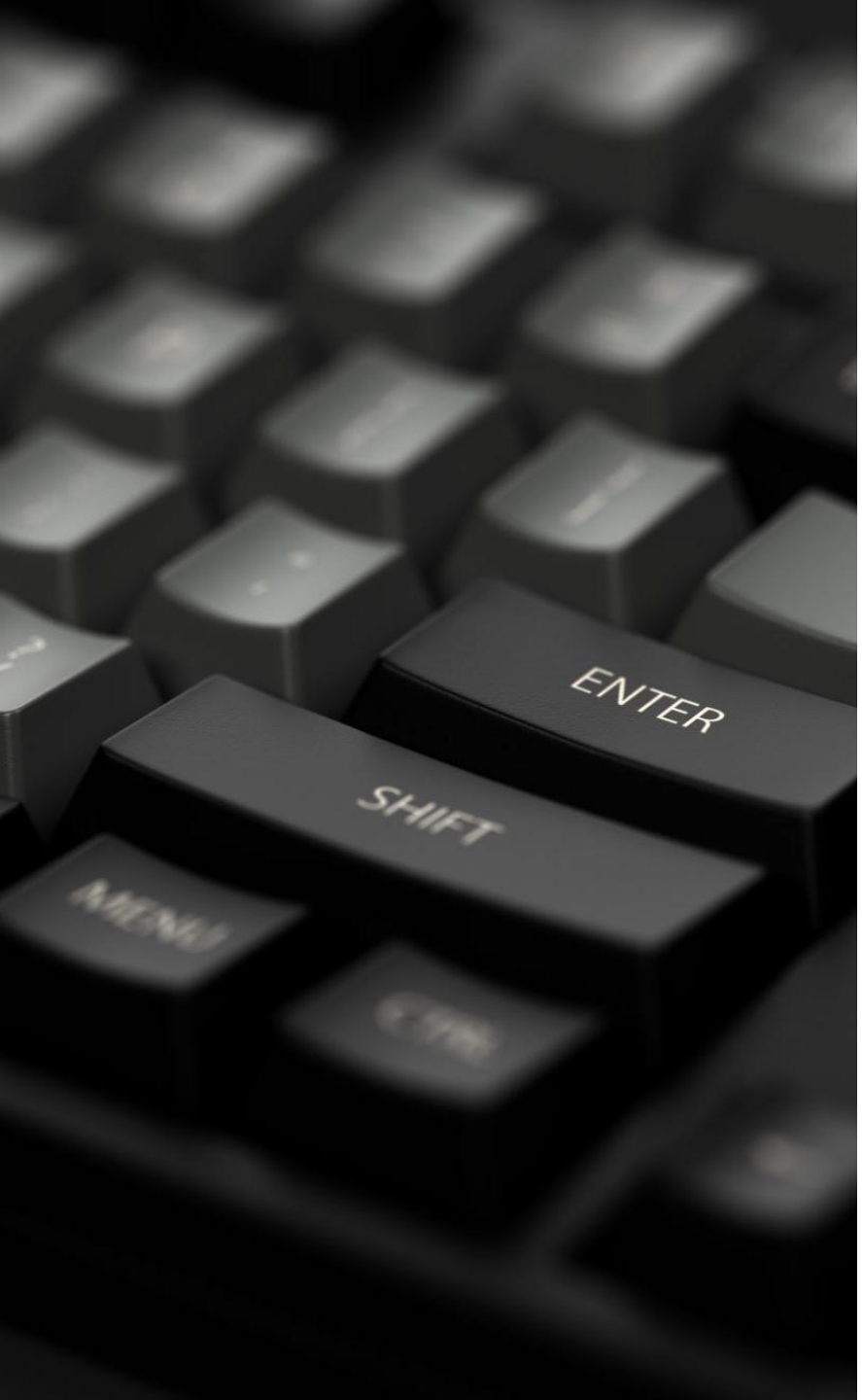
Let us practice these concepts with a second activity.

Check the notebook

Activity 2 - Ballistics of an angry bird.ipynb

In this notebook, you will have to write a single function,

- which computes the distance at which an angry bird will be landing,
- depending on a given initial angle,
- and an initial speed.



Conclusion (Chapter 3)

- What are the **None** type and the **pass** keyword?
- What is the **Boolean** type?
- **Boolean quiz** for practice.
- **Advanced concepts on Booleans**.
- What are **functions** in Python and how to use them?
- What are **memory states in functions**?

More activities for you to practice!

Activity 3: Distance to Point of Interest, flat earth version

In several video games, you can track the position of a point of interest and display the distance between the current player's position, defined as (x_p, y_p); and the point of interest (PoI) located at the position (x_t, y_t).

Write a function **distance_to_poi()** which receives 4 input parameters (x_p, y_p, x_t, y_t) and returns the distance d, in meters, between the player's position (x_p, y_p) and the PoI position (x_t, y_t).

More activities for you to practice!

Activity 4: Distance to Point of Interest, spherical earth version

Same as Activity 3, but we now consider that the map model is no longer flat, but spherical.

Instead of (x, y) coordinates, we will use latitude and longitude coordinates.

The formula for computing the distance changes, into something slightly more difficult, which will require to import a few mathematical functions from the numpy library.

More activities for you to practice!

Activity 5: Is this triangle a right-angle triangle?

Consider a triangle with three lengths values a, b, and c (with c being the largest of all three values).

Write a function, which receives all three values and returns a Boolean, with value:

- True, if the triangle is a right-angle triangle.
- False otherwise.

The function should return False, if the values a, b and c passed are such that c is not the largest of all three values.

*Do not use **if** statements just yet!*

Activity 6: What color is the square?

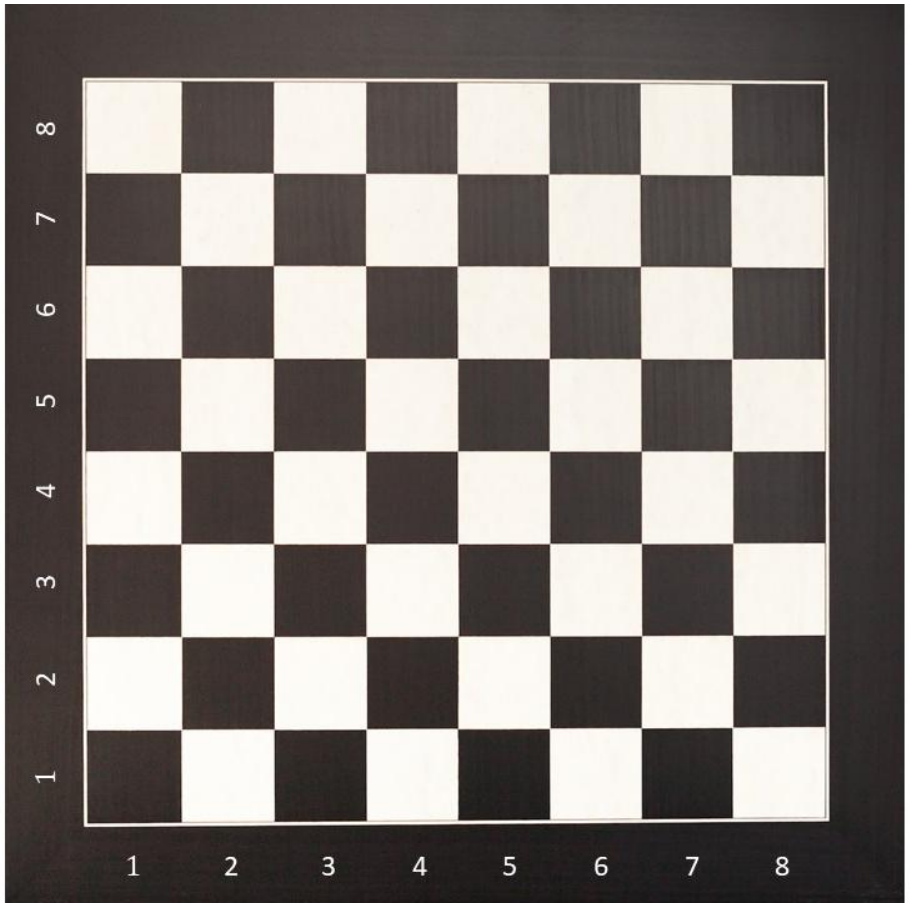
Activity 6: What color is the square?

In this notebook, you will have to write several functions:

- To collect user's inputs on rows and columns indexes,
- To check and print if the square is black or white.

Later on, you should assemble them in a nice main() function!

*Do not use **if** statements just yet!*

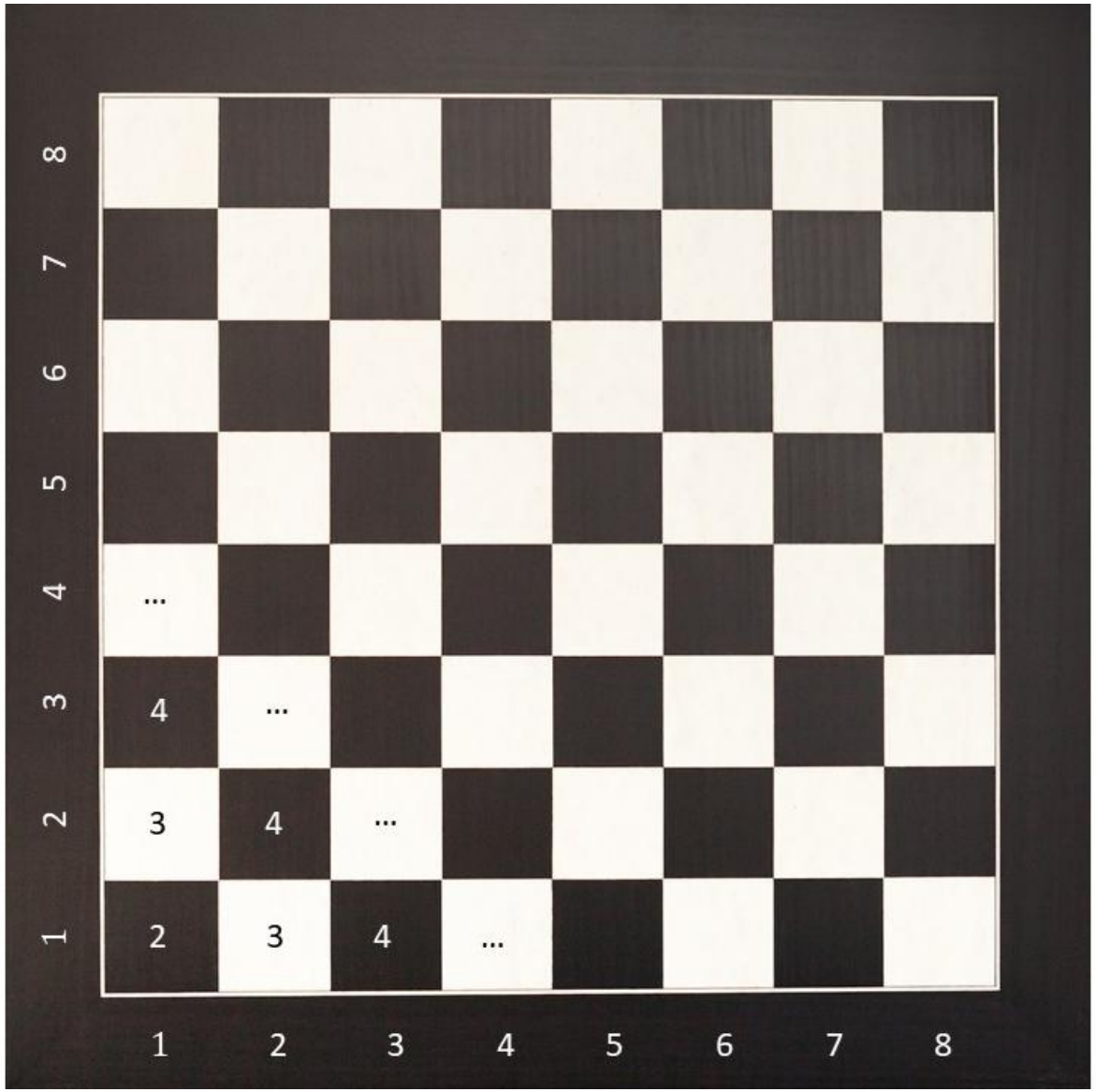


Activity 6: Hint

Activity 6: What color is the square?

- **Hint:** what do I get if I sum the row and column indexes for all squares?

- Do you recognize a pattern?



More activities for you to practice!

- As usual, if needed, some extra Practice/Challenges are available! *(Feel free to explore on your own!)*

- **Activity 7: Guess the card game (used to be an in-class activity)**

Same as Activity 1 (guess the number), but we now consider that the player must guess a card (color and values) instead of just a number.

You will have to write a few subfunctions and a main() function.

*Do not use **if** statements just yet!*