ILP 2020 – W1S3
None type, Boolean type and functions

Matthieu DE MARI – Singapore University of Technology and Design

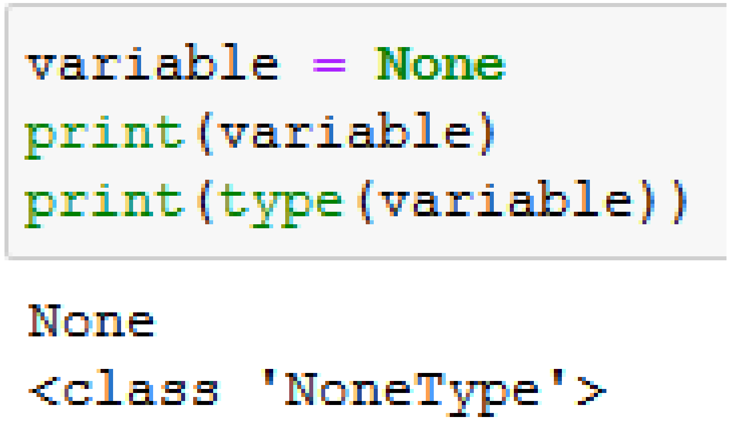
SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Outline (Week1, Session3 – W1S3)

- The None type
- The Boolean type
- Boolean quiz
- Advanced concepts on Booleans
- Functions
- 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 **block a variable name** in advance, even though **no value has yet been assigned** to it.

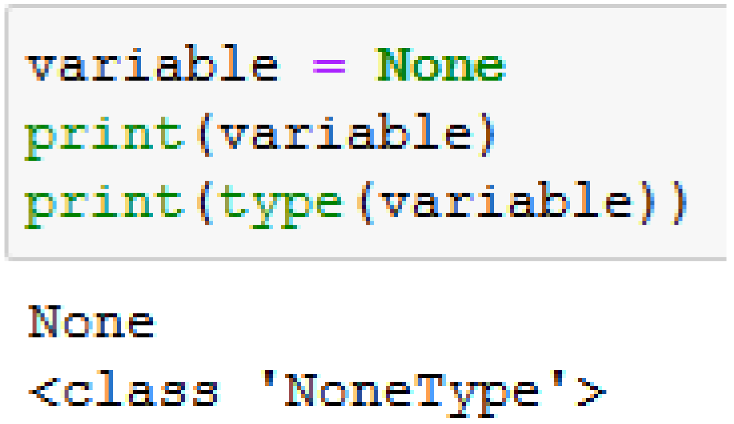


```python
variable = None
print(variable)
print(type(variable))
```

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

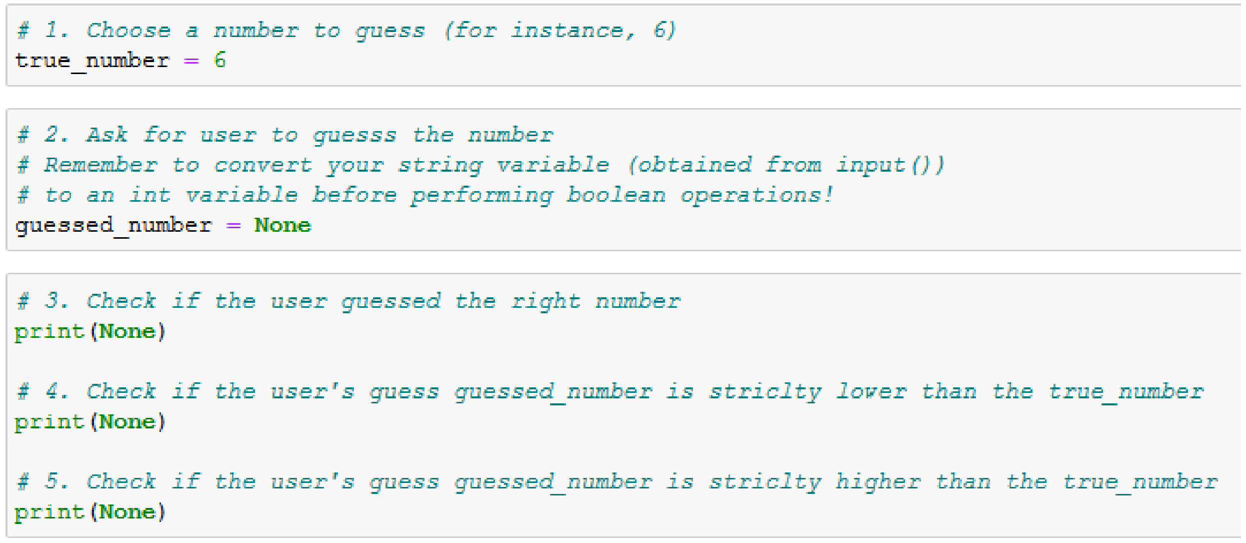
The None type

- **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 **block a variable name** in advance, even though **no value has yet been assigned** to it.
- You will see me use it in some activities to “hint” on some variables or functions you should be using.



```python
variable = None
print(variable)
print(type(variable))
```

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



```python
# 1. Choose a number to guess (for instance, 6)
true_number = 6
```

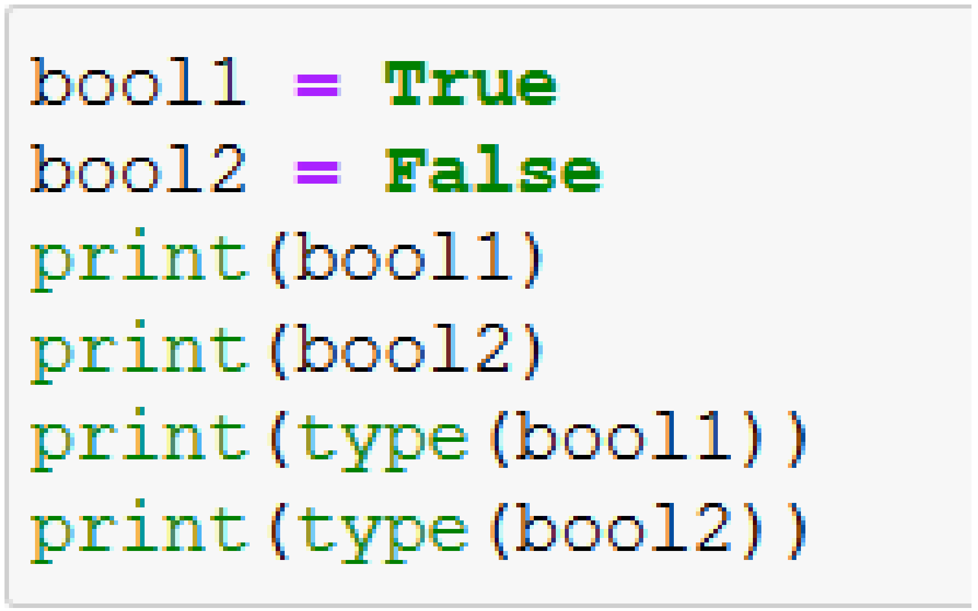
```python
# 2. Ask for user to guesss the number
# Remember to convert your string variable (obtained from input())
# to an int variable before performing boolean operations!
guessed_number = None
```

```python
# 3. Check if the user guessed the right number
print(None)

# 4. Check if the user's guess guessed_number is striclty lower than the true_number
print(None)

# 5. Check if the user's guess guessed_number is striclty higher than the true_number
print(None)
```

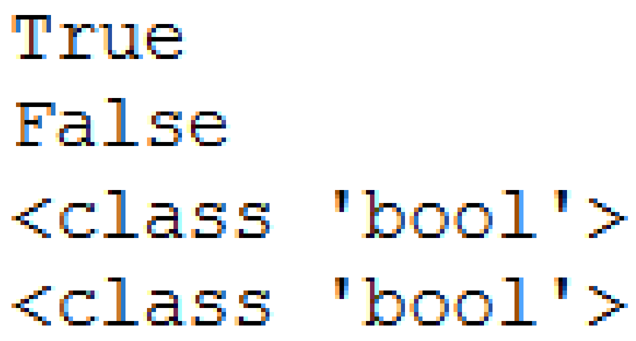
The boolean type

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



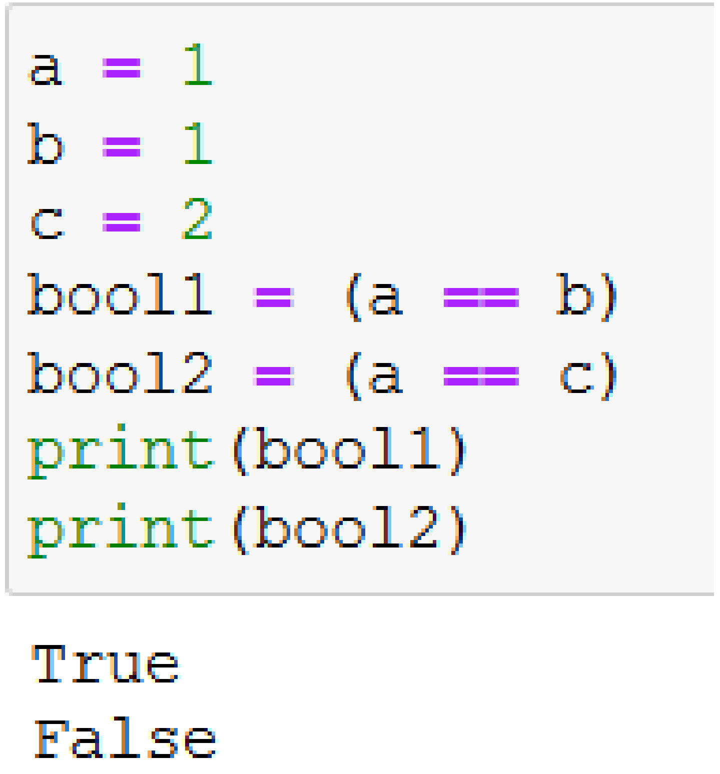
```
bool1 = True
bool2 = False
print(bool1)
print(bool2)
print(type(bool1))
print(type(bool2))
```



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

Boolean comparisons

- **Definition (the == operator):**
  The == operator is used to check if two variables have identical values.
  The result of this operation is a Boolean, with value
  - **True**, if both variables have **identical values**;
  - and **False**, **otherwise**.

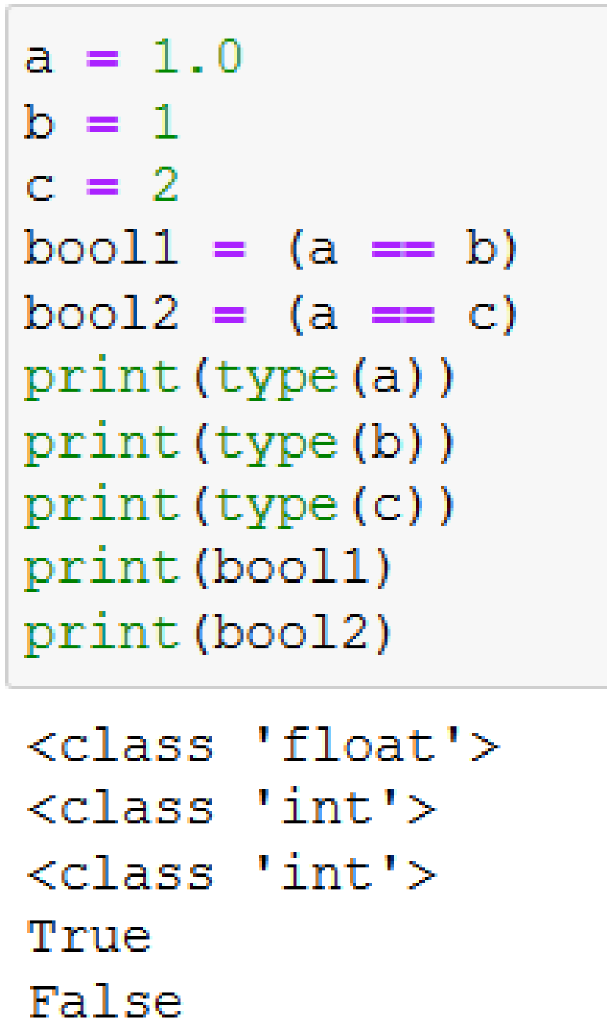


```
a = 1
b = 1
c = 2
bool1 = (a == b)
bool2 = (a == c)
print(bool1)
print(bool2)
```

```
True
False
```

Boolean comparisons

- **Definition (the == operator):** The == operator is used to check if two variables have identical values.
  The result of this operation is a Boolean, with value
  - **True**, if both variables have **identical values**;
  - and **False**, **otherwise**.
- **Note:** Also, works if variables have mixed int/float types.

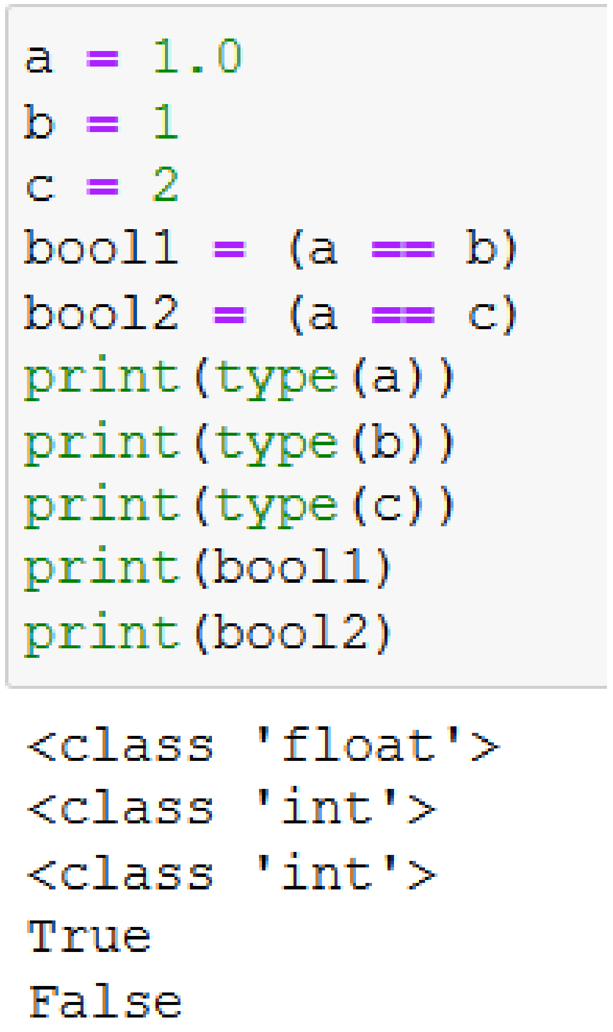


```
a = 1.0
b = 1
c = 2
bool1 = (a == b)
bool2 = (a == c)
print(type(a))
print(type(b))
print(type(c))
print(bool1)
print(bool2)
```

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

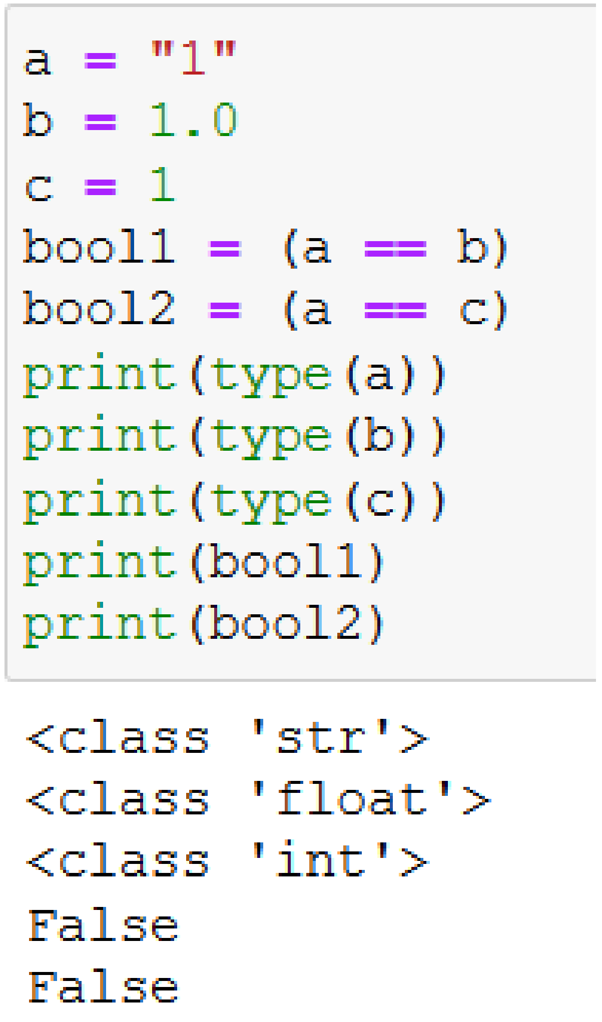
Boolean comparisons

- **Definition (the == operator):** The == operator is used to check if two variables have identical values.
  The result of this operation is a Boolean, with value
  - **True**, if both variables have **identical values**;
  - and **False**, **otherwise**.
- **Note:** Also, works if variables have mixed int/float types.
  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
```

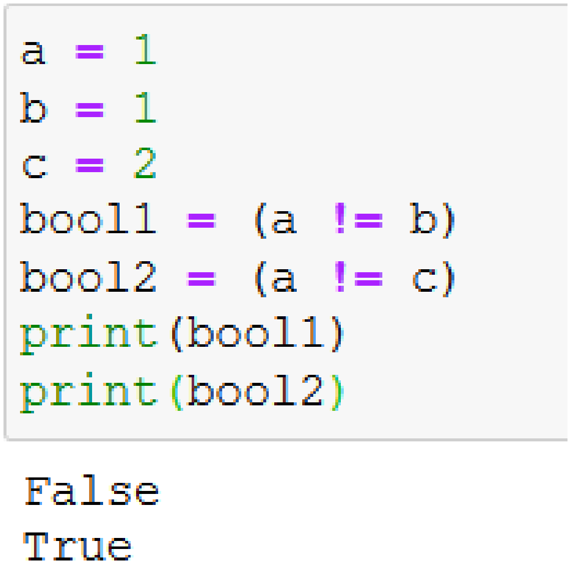


```
a = "1"
b = 1.0
c = 1
bool1 = (a == b)
bool2 = (a == c)
print(type(a))
print(type(b))
print(type(c))
print(bool1)
print(bool2)
```

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

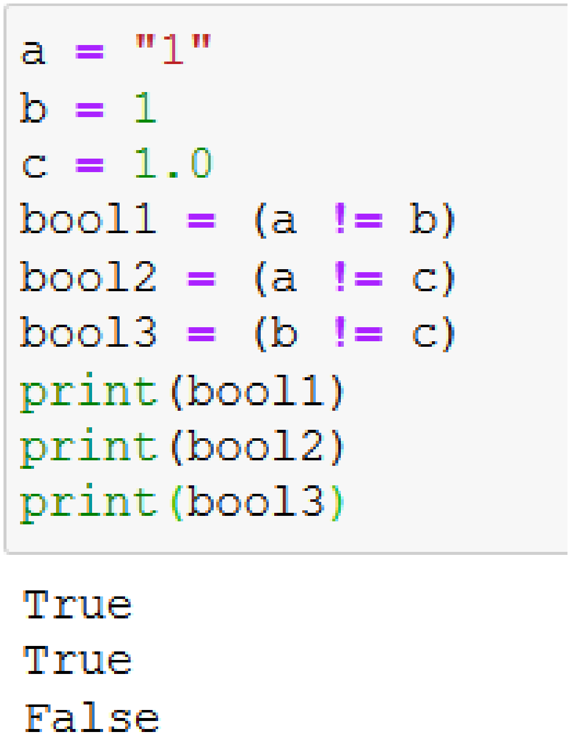
Boolean comparisons

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



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

```
False
True
```



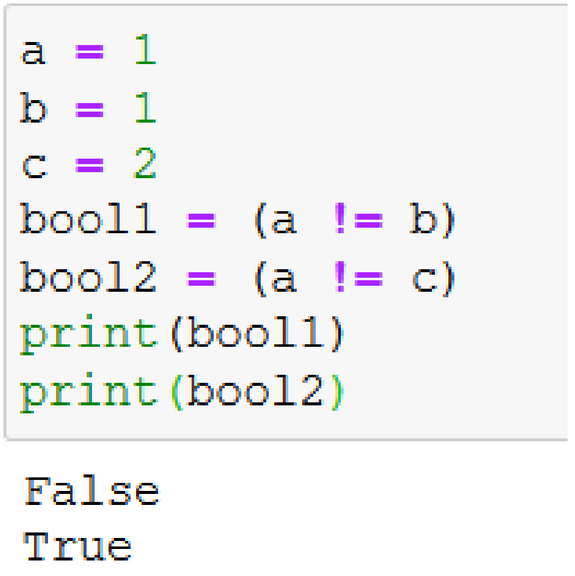
```python
a = "1"
b = 1
c = 1.0
bool1 = (a != b)
bool2 = (a != c)
bool3 = (b != c)
print(bool1)
print(bool2)
print(bool3)
```

```
True
True
False
```

Boolean comparisons

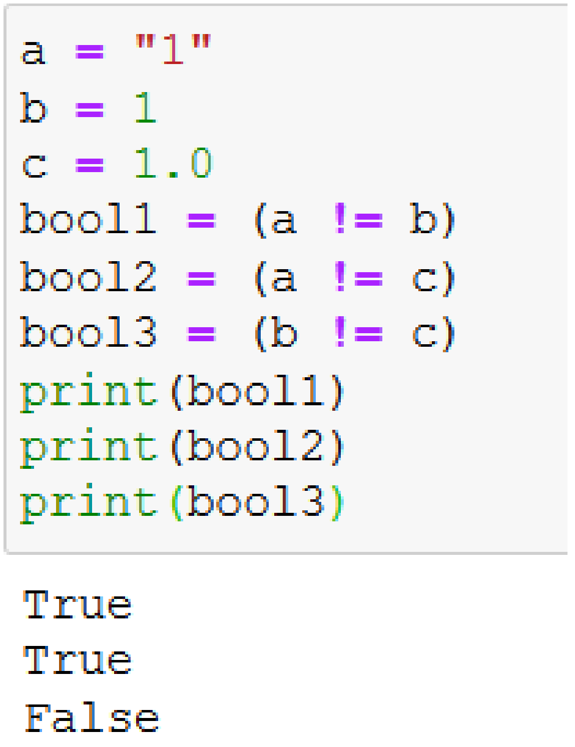
- Following the same logic as the == operator, we can define some additional operators.

- !=: if variables have **different values**.



```
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)
```

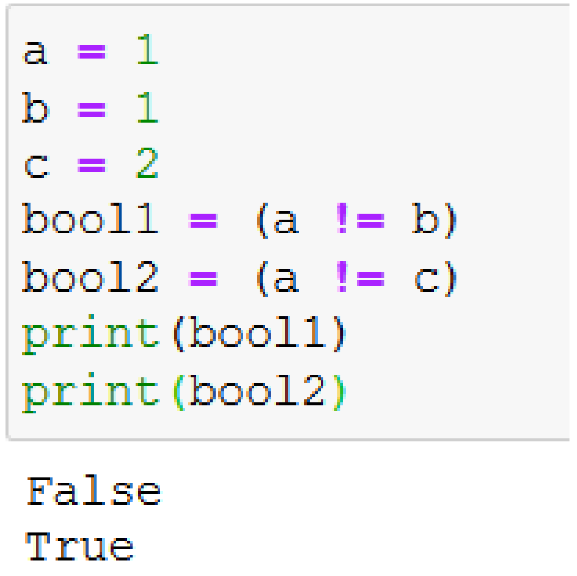
```
True
True
False
```

Boolean comparisons

- Following the same logic as the **==** operator, we can define some additional operators.

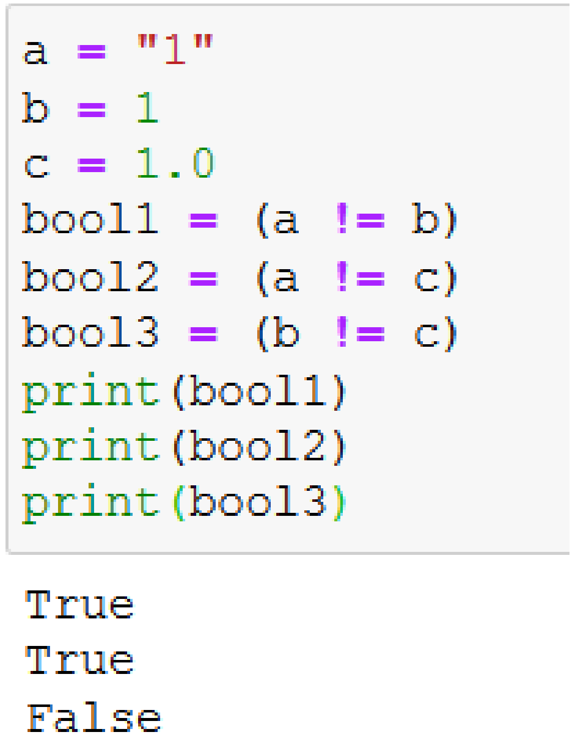
- **!=**: if variables have **different values**.

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



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

```
False
True
```



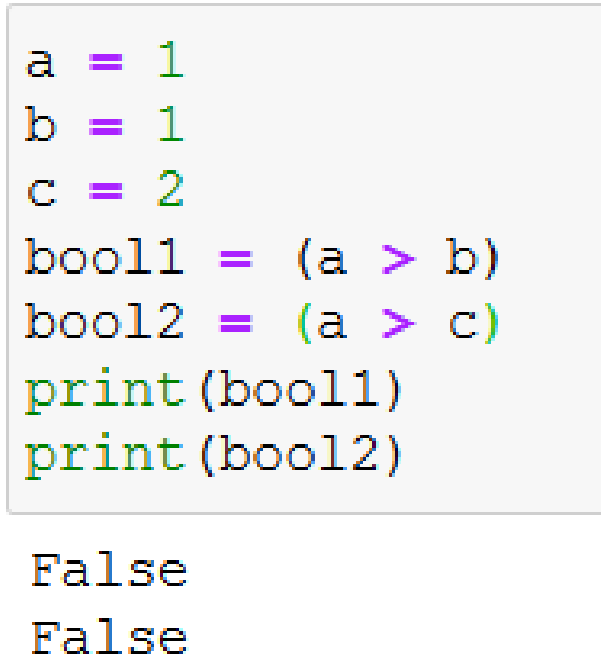
```python
a = "1"
b = 1
c = 1.0
bool1 = (a != b)
bool2 = (a != c)
bool3 = (b != c)
print(bool1)
print(bool2)
print(bool3)
```

```
True
True
False
```

Boolean comparisons

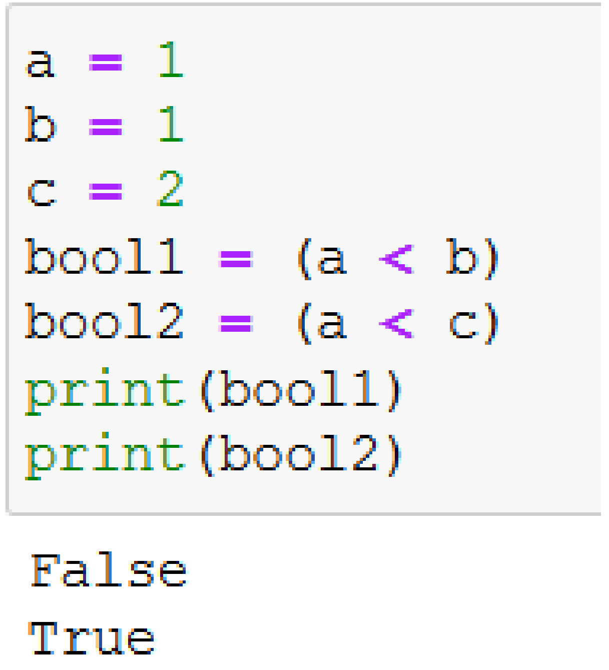
Similarly,

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



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

```
False
False
```



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

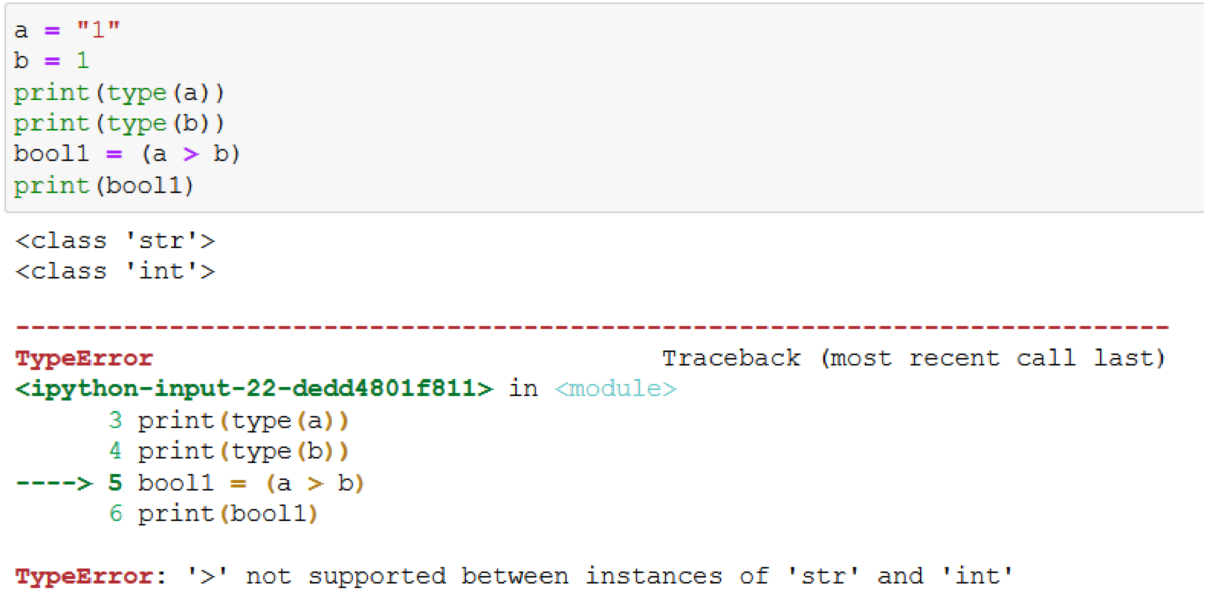
```
False
True
```

Boolean comparisons

Similarly,

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

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



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

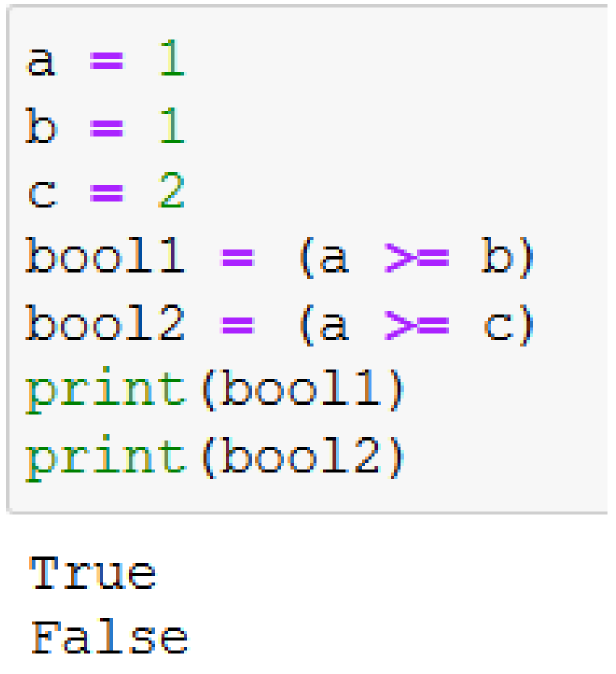
```
<class 'str'>
<class 'int'>

---------------------------------------------------------------------------
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,

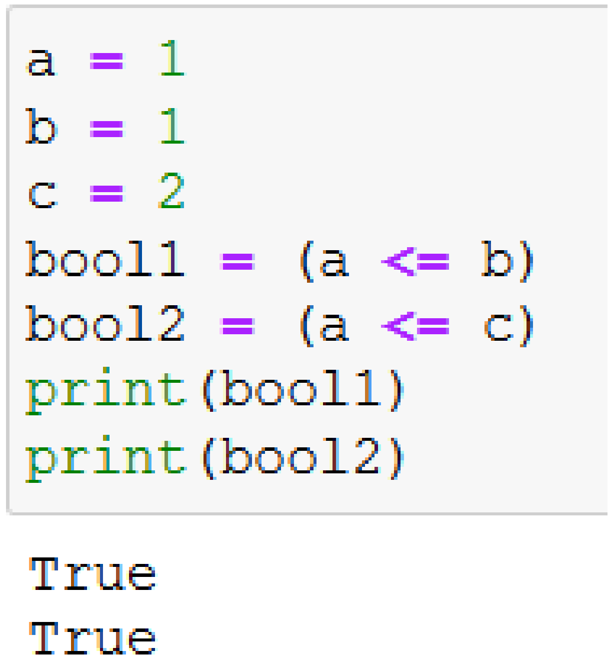
- **>=:** if variable on the left-hand side has a **higher or equal numerical value**, than the one on the right-hand side.
- **<=:** same as >=, but checking for **lower or equal numerical value**.

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



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

```
True
False
```



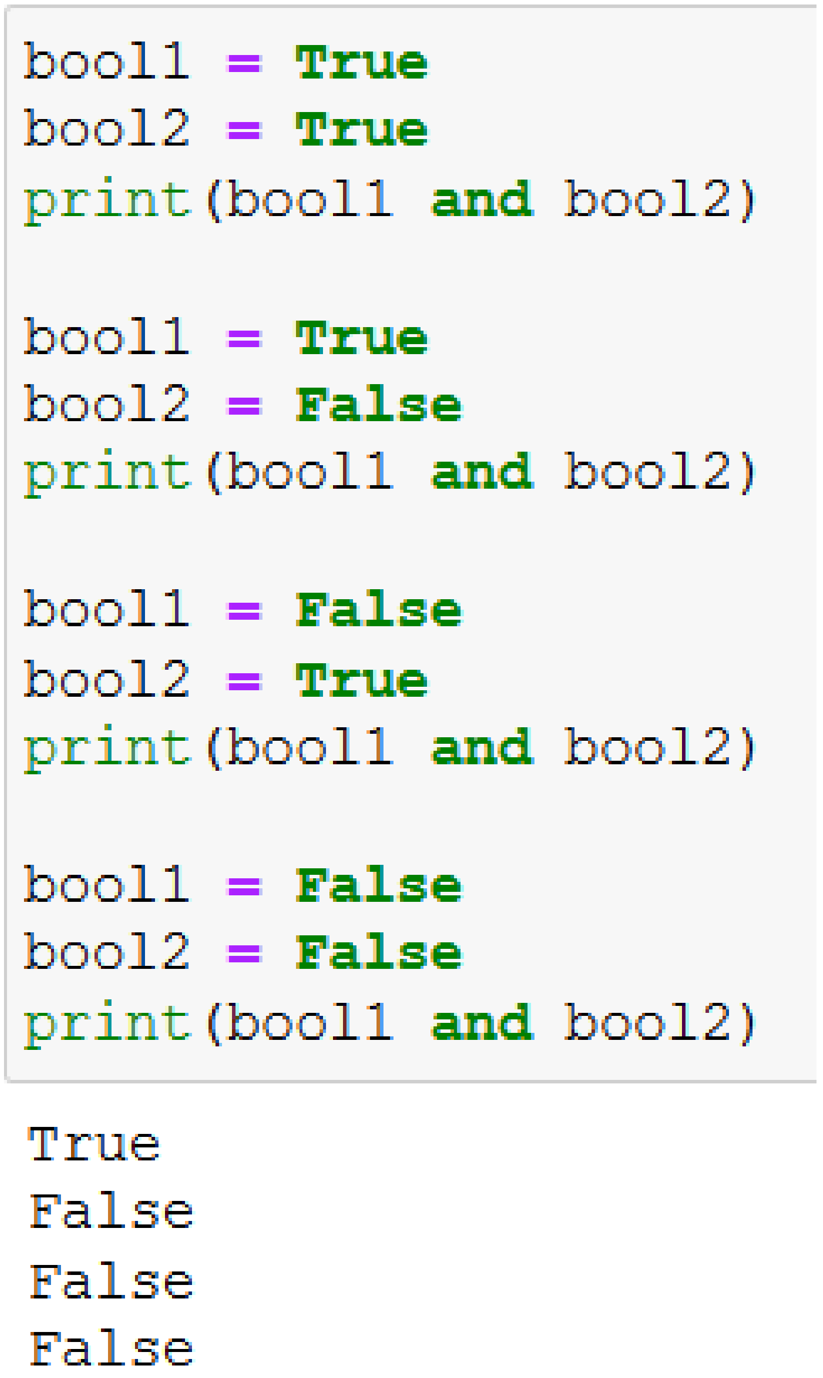
```
a = 1
b = 1
c = 2
bool1 = (a <= b)
bool2 = (a <= c)
print(bool1)
print(bool2)
```

```
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.



```
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)
```

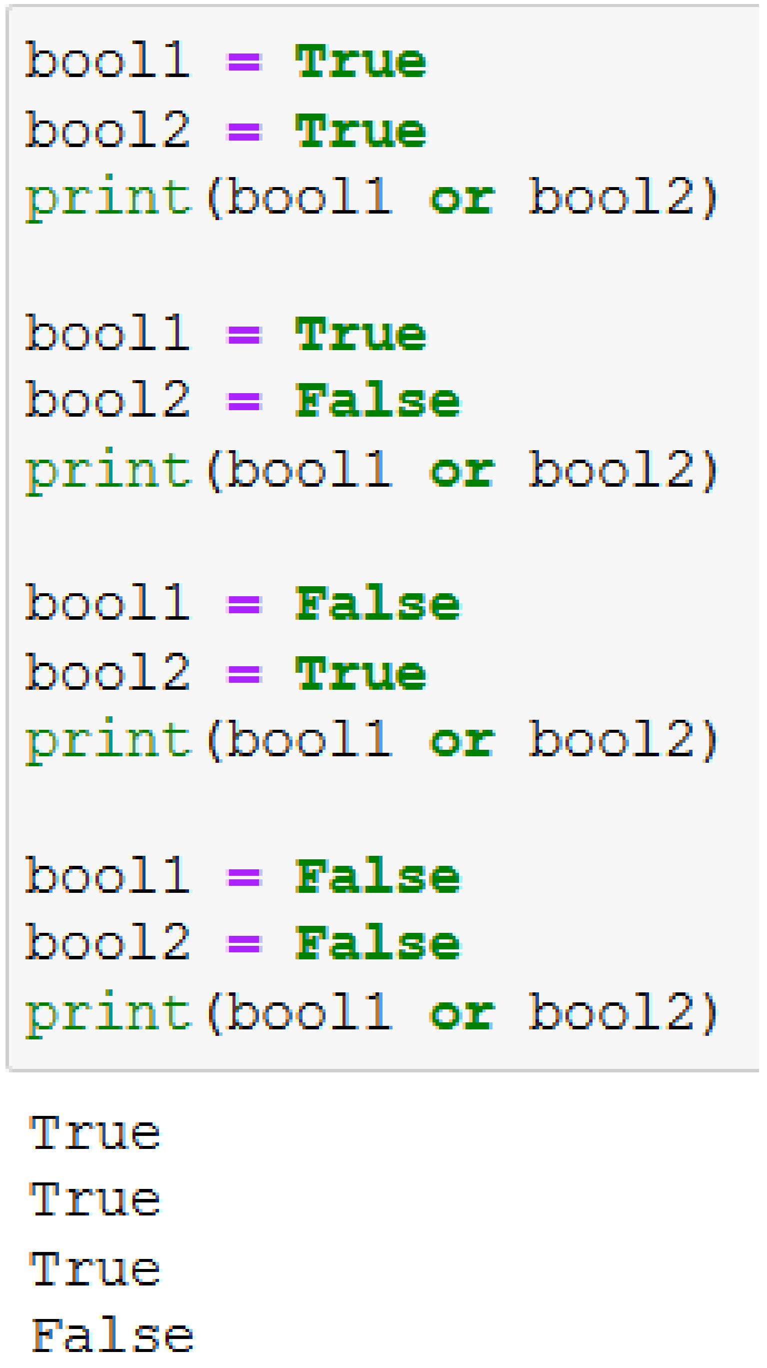
```
True
False
False
False
```

Boolean combinations: and, or, not

Similarly,

- **Definition (the or operator):** 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.



```
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

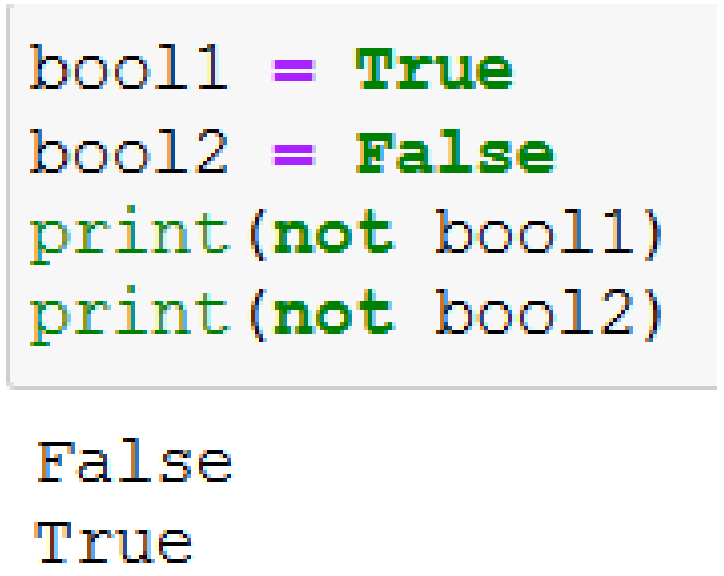
Similarly,

- **Definition (the or operator):** 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.

Boolean combinations: and, or, not

- **Definition (the not operator):** The not operator returns a boolean with **opposite value.**
    - **not True** is **False**;
    - **not False** is **True**.



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

```
False
True
```

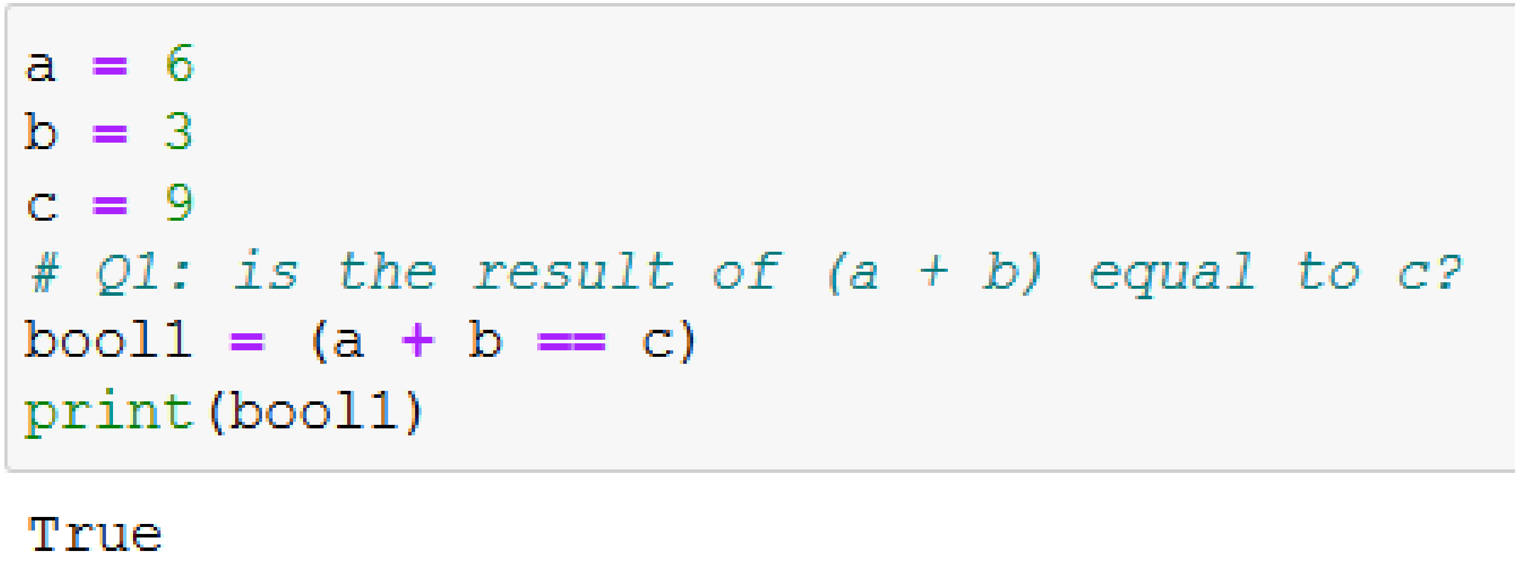
Boolean type: practice quiz!

Let us practice a bit with a quick quiz on Booleans!

https://docs.google.com/forms/d/e/1FAIpQLSdvkohaamF-MqvH_1zTUpmR__2mva8VVRYpHeQDBLPWd0V0hA/viewform?usp=sf_link

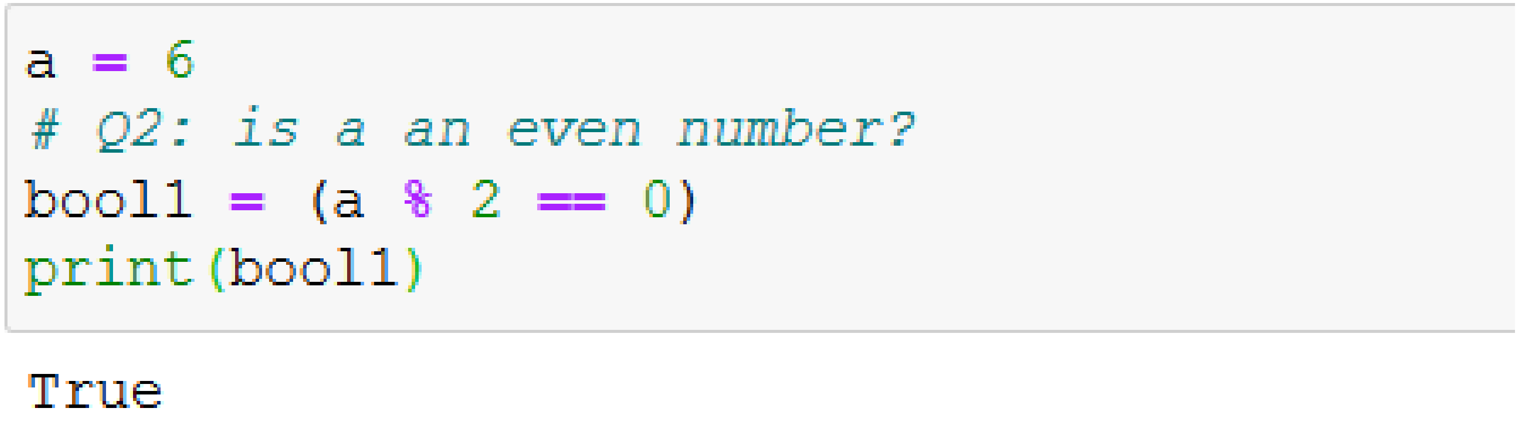
(Link will be posted on Zoom)

Boolean type: practice quiz! (answers)



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

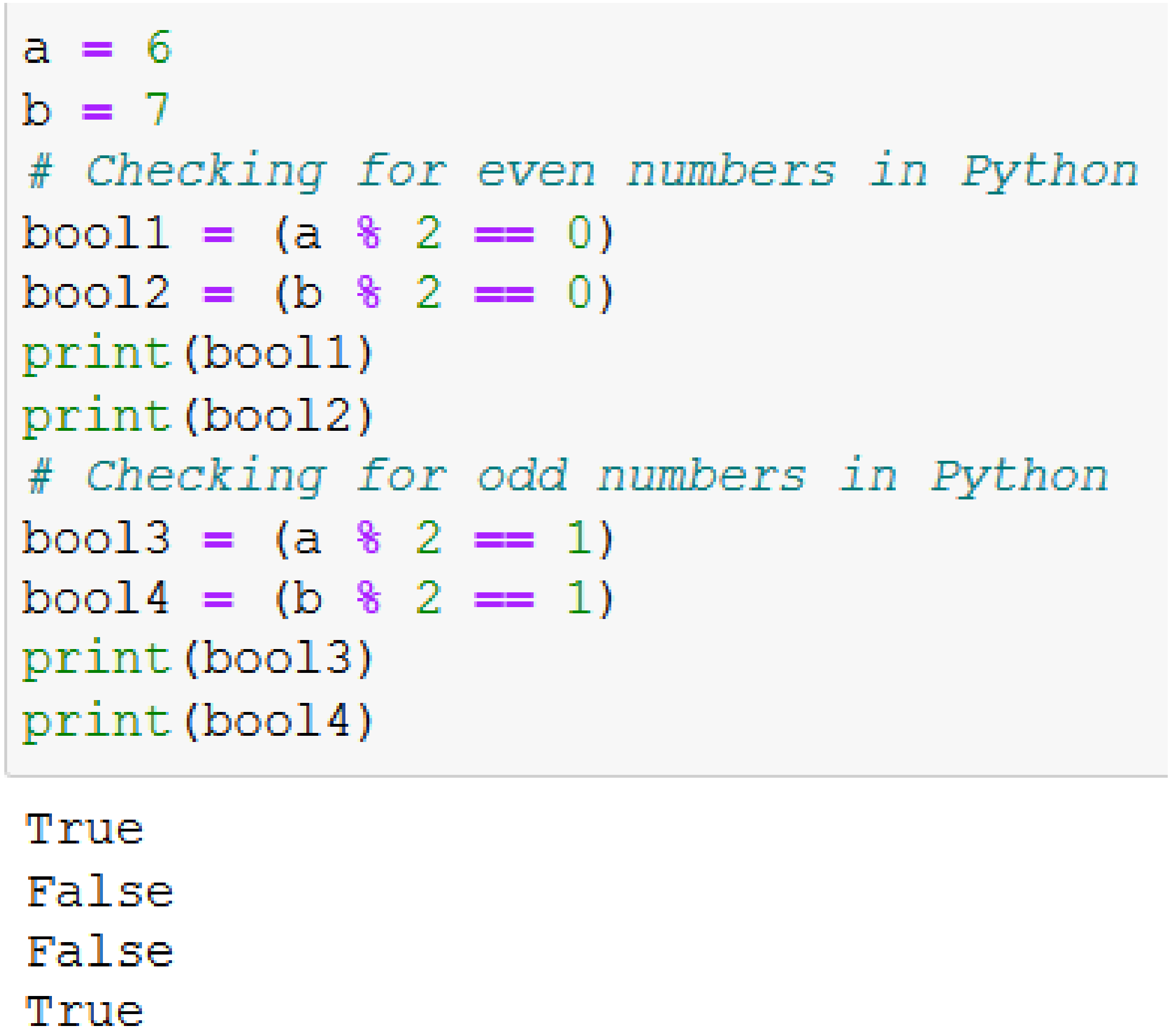
```
True
```



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

```
True
```

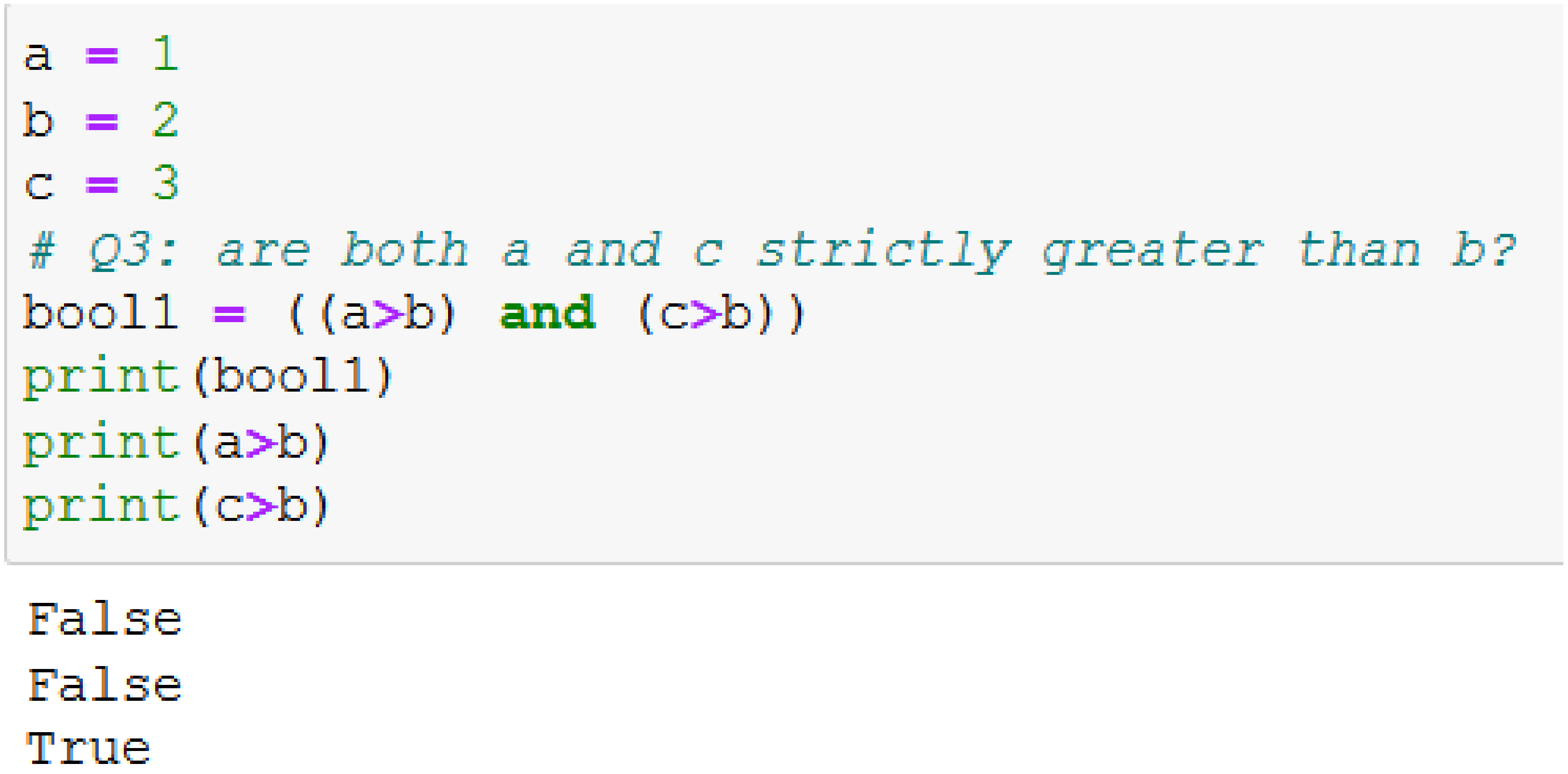
Boolean type: practice quiz! (answers)



```python
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
```

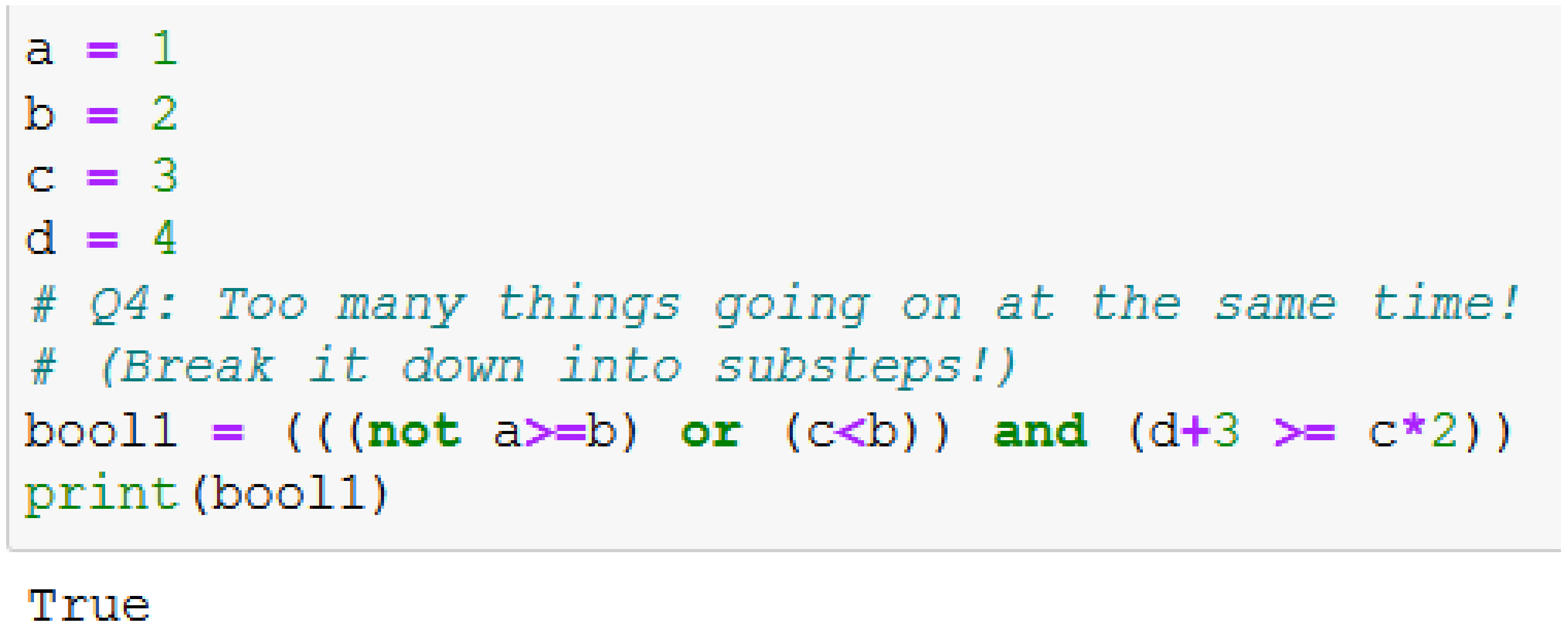
Boolean type: practice quiz! (answers)



```
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
```

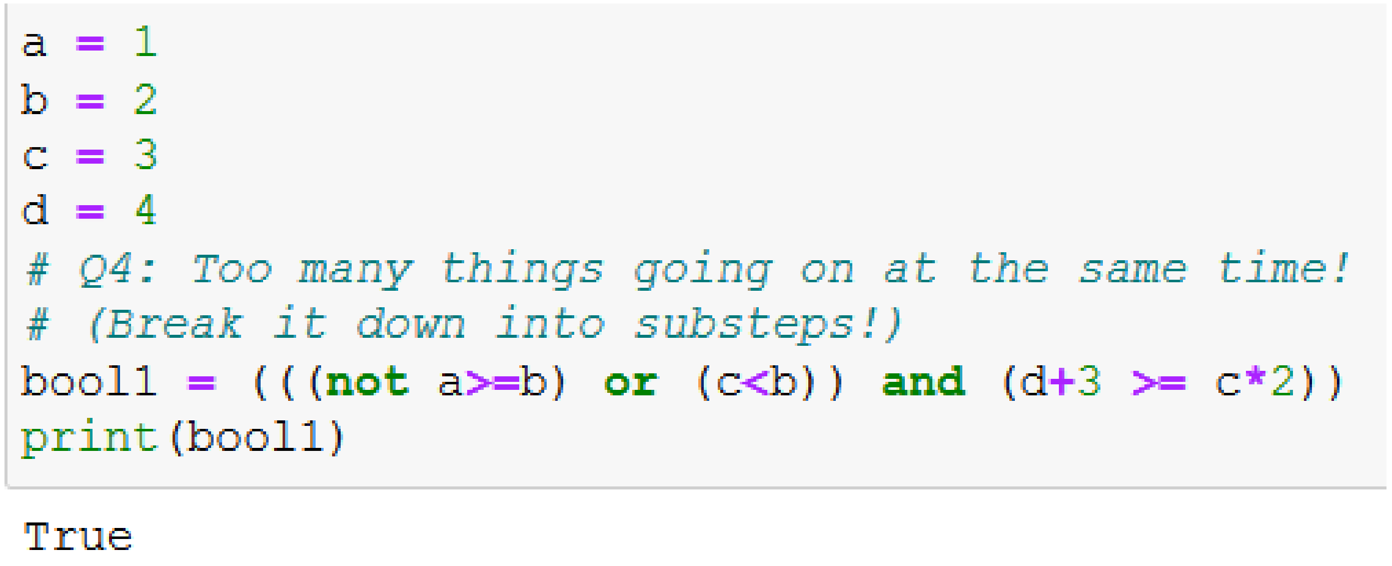
Boolean type: practice quiz! (answers)



```python
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)
```

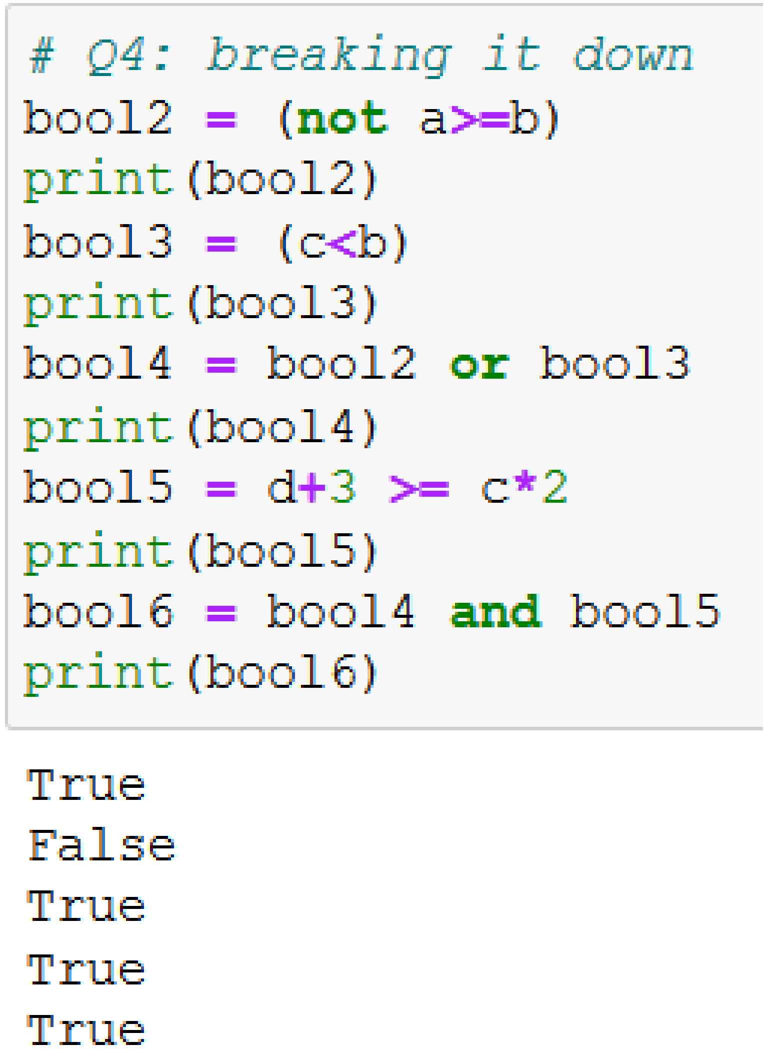
```
True
```

Boolean type: practice quiz! (answers)



```python
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
```



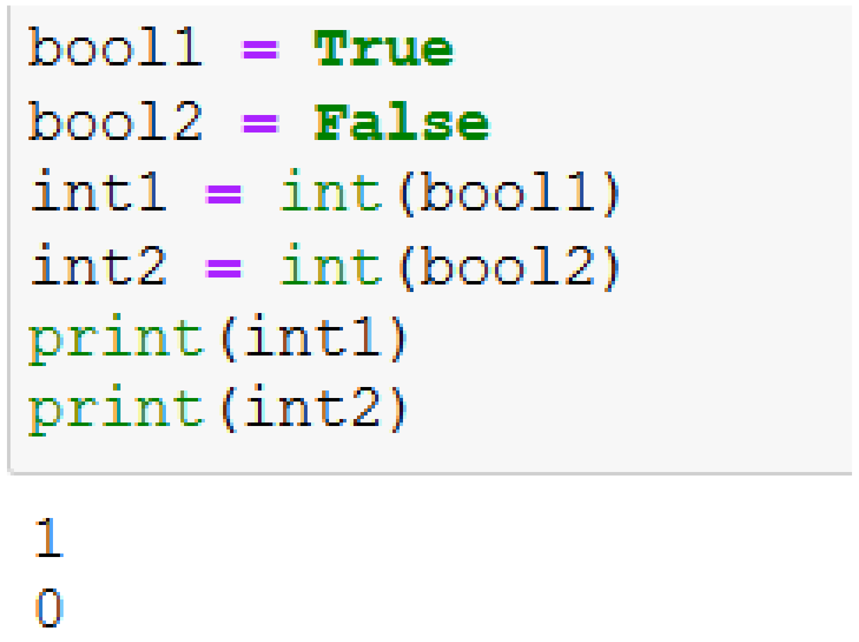
```python
# Q4: breaking it down
bool2 = (not a>=b)
print(bool2)
bool3 = (c<b)
print(bool3)
bool4 = bool2 or bool3
print(bool4)
bool5 = d+3 >= c*2
print(bool5)
bool6 = bool4 and bool5
print(bool6)
```

```
True
False
True
True
True
```

Boolean conversion

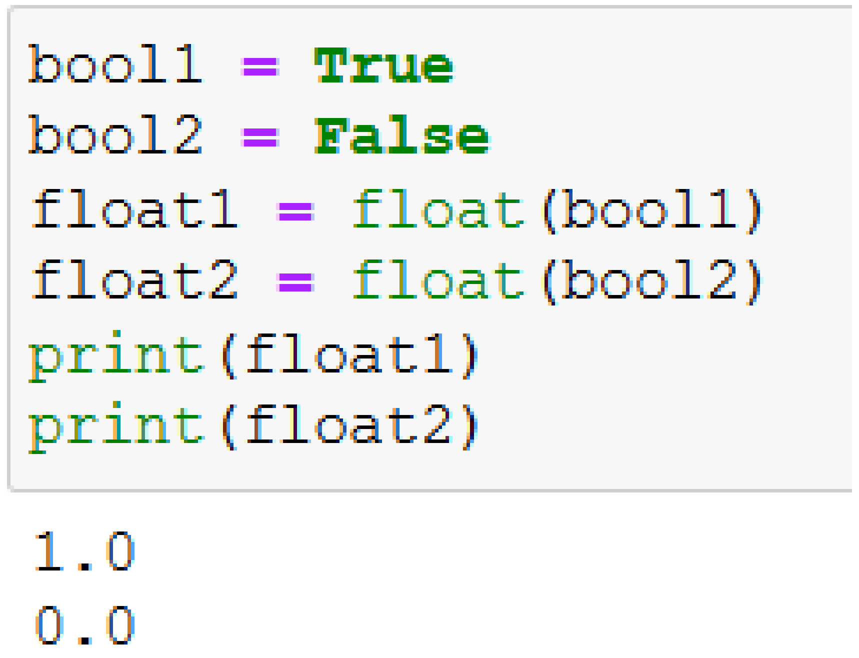
Bool -> Int/Float: You can convert a Boolean into an int/float number.

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



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

```
1
0
```



```python
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).**

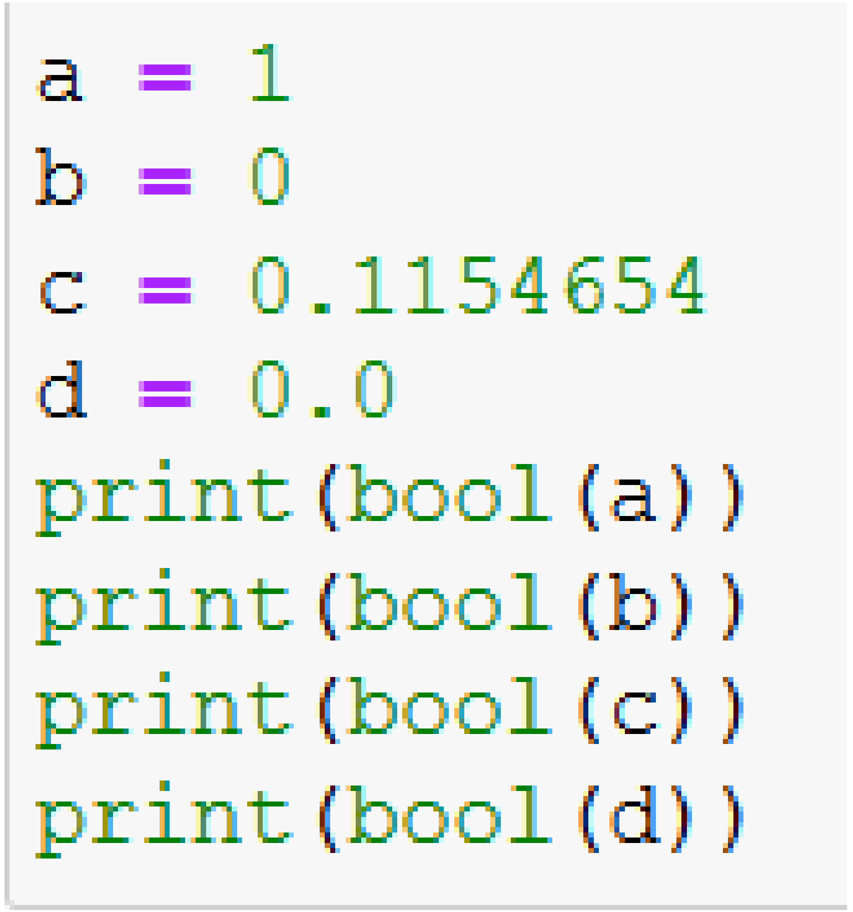
- **Fun fact:** that is reason behind the on/off symbols.

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
```

Activity 1: Guess the number game!

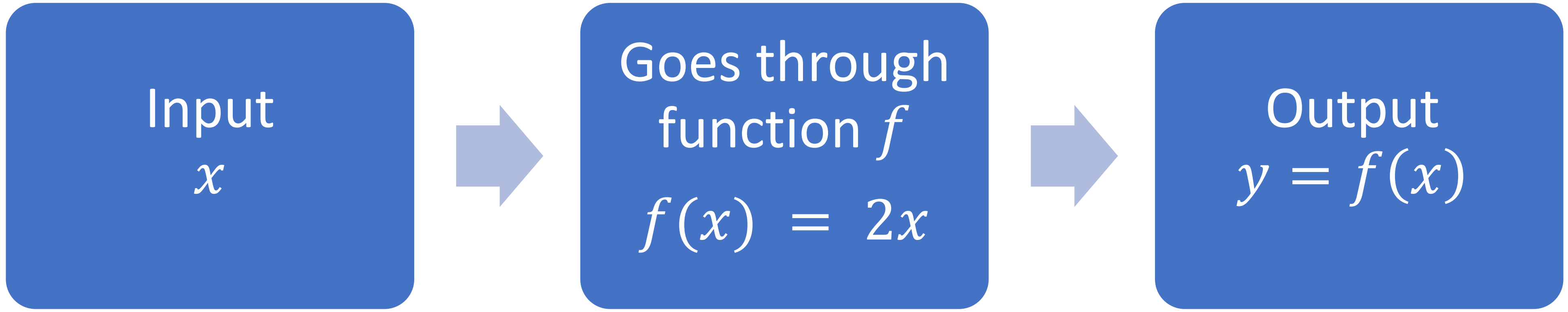
Let us play a bit with the concepts, with a first activity

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

Function objects

- In mathematics, we often like to define **functions**, in the form

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



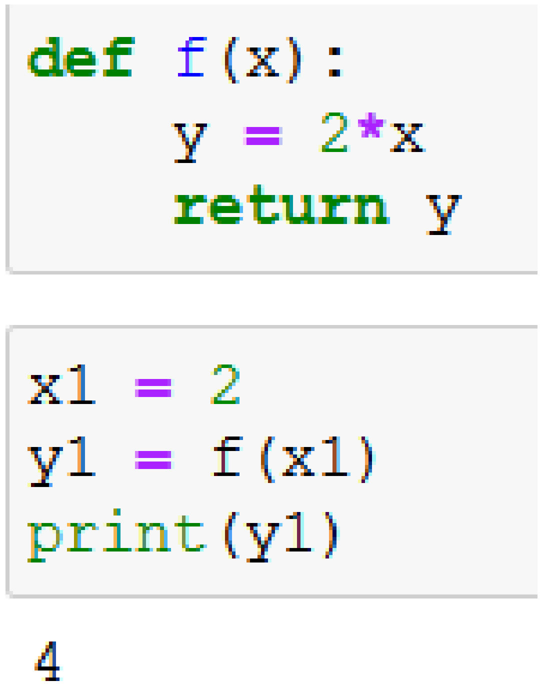
- The same can be done in Python, by creating a function.

Function formatting in Python

- **Definition (Python functions):**
  A Python **function** is a block of code which only runs when it is **called**.

  You can **pass data**, known as **parameters** or **input values**, into a function.

  A function can **return data**, as a **result** or **output values**.



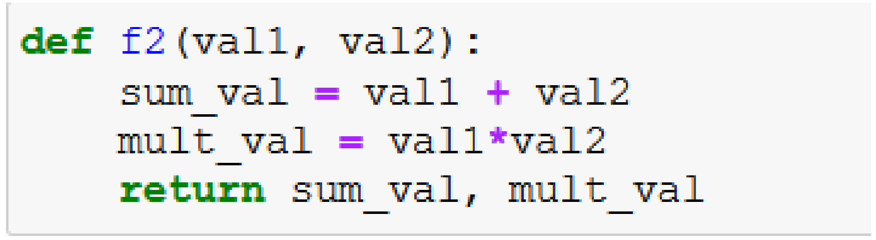
```
def f(x):
    y = 2*x
    return y
```

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

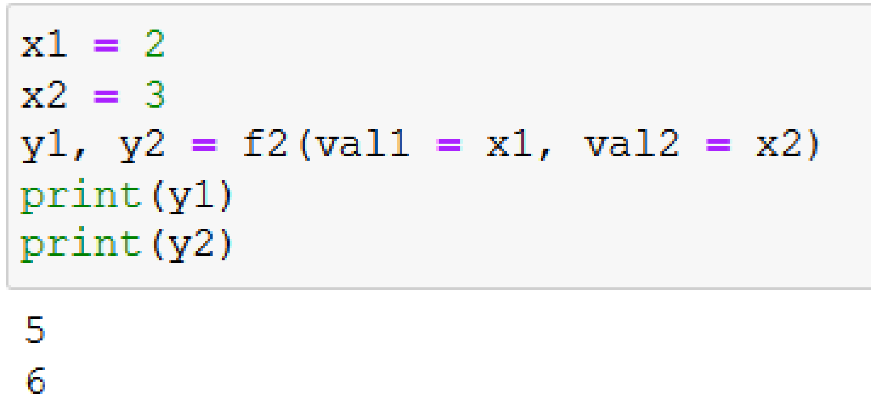
```
4
```

Function formatting in Python

- You can define a function, with a **def** statement.
- Immediately after **def**, type the **function's name**.

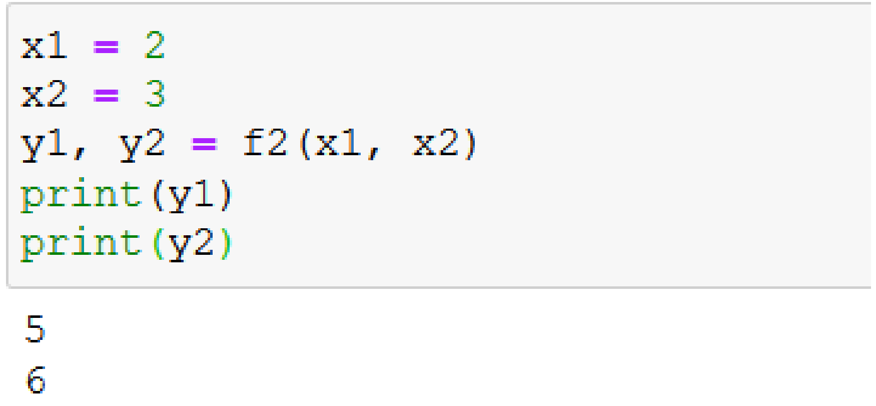


```
def f2(val1, val2):
    sum_val = val1 + val2
    mult_val = val1*val2
    return sum_val, mult_val
```



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

```
5
6
```



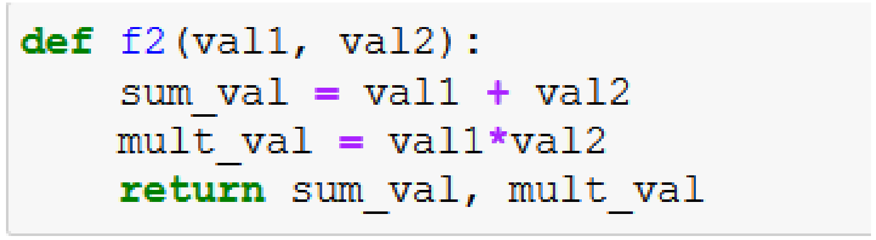
```
x1 = 2
x2 = 3
y1, y2 = f2(x1, x2)
print(y1)
print(y2)
```

```
5
6
```

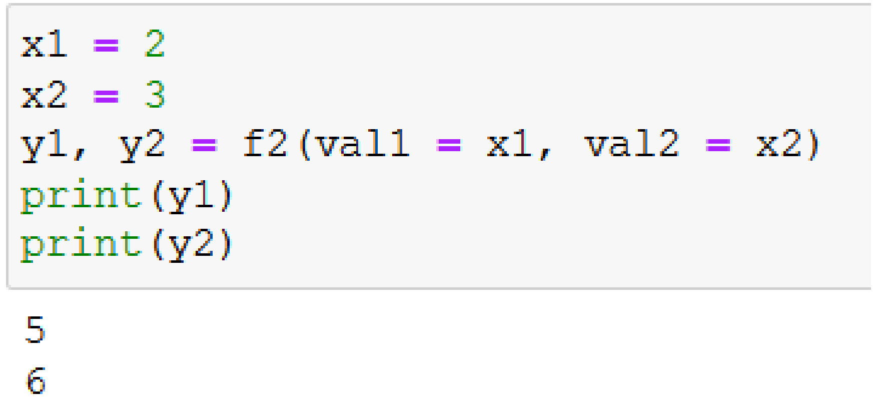
Function formatting in Python

- You can define a function, with a **def** statement.
- Immediately after **def**, type the **function's name**.

- **Between parentheses**, after the function's name, type **input values/parameters**.
- You may have multiple inputs, separated with commas.

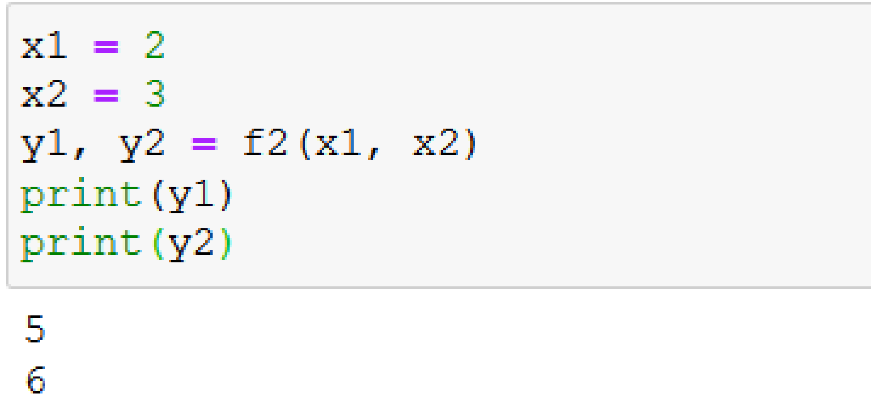


```python
def f2(val1, val2):
    sum_val = val1 + val2
    mult_val = val1*val2
    return sum_val, mult_val
```



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

```
5
6
```



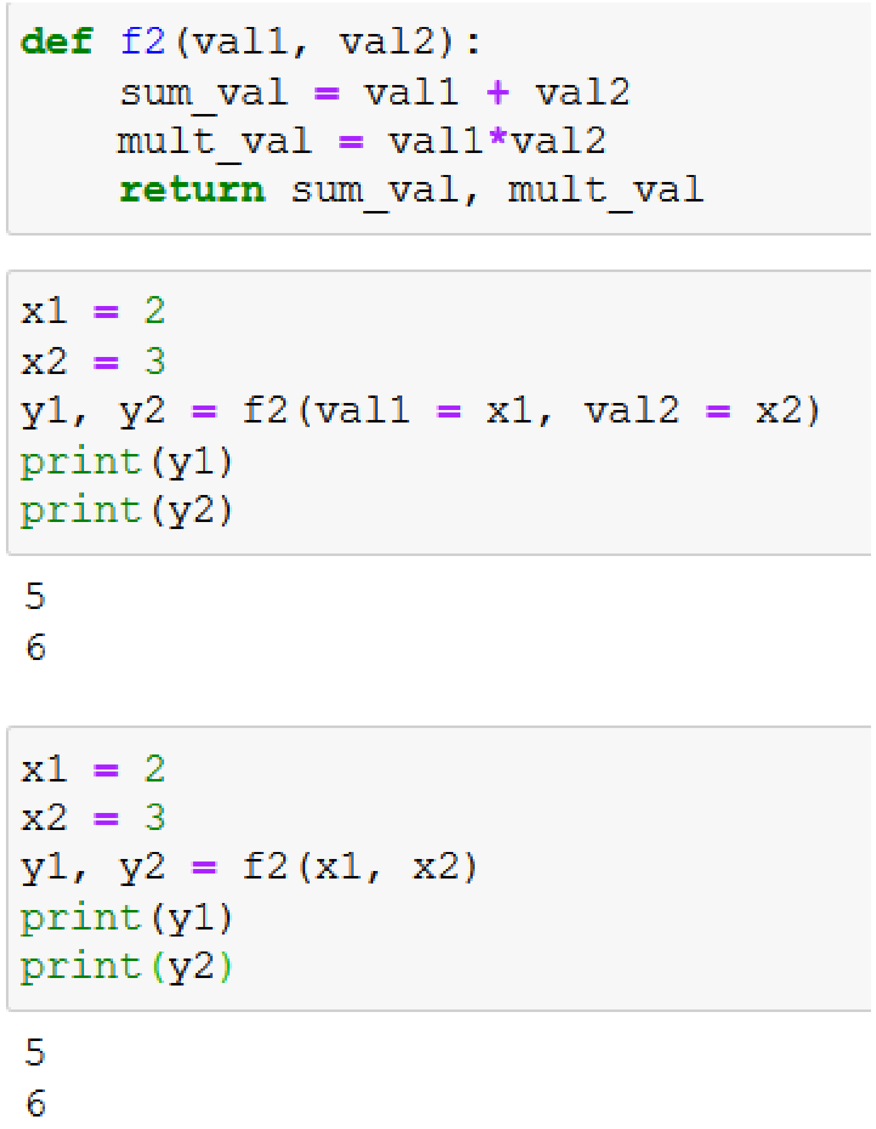
```python
x1 = 2
x2 = 3
y1, y2 = f2(x1, x2)
print(y1)
print(y2)
```

```
5
6
```

Function formatting in Python

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

- You may have multiple outputs, as well as inputs, separated with commas.



```
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
```

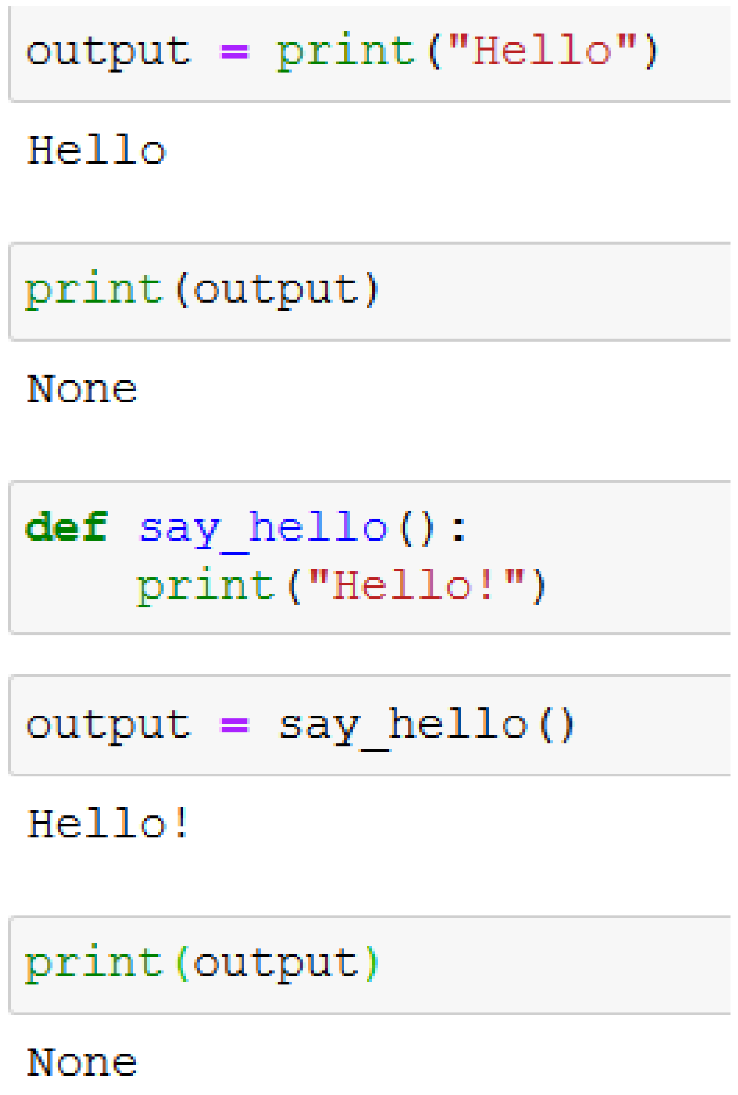
```
x1 = 2
x2 = 3
y1, y2 = f2(x1, x2)
print(y1)
print(y2)
```

```
5
6
```

Function formatting in Python

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

- You may have multiple outputs, as well as inputs, separated with commas.
- **Note:** Your function may also not **return** anything (that is the case for the **print()** function!)



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

```
Hello
```

```
print(output)
```

```
None
```

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

```
output = say_hello()
```

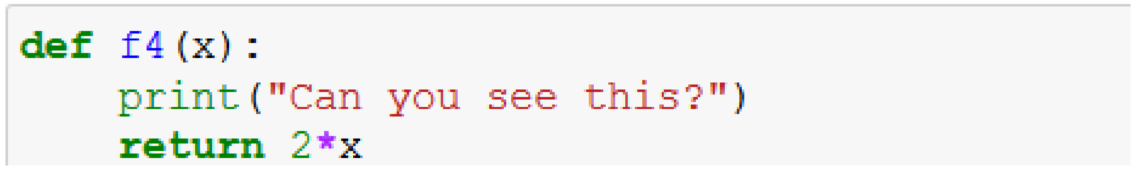
```
Hello!
```

```
print(output)
```

```
None
```

Function formatting in Python

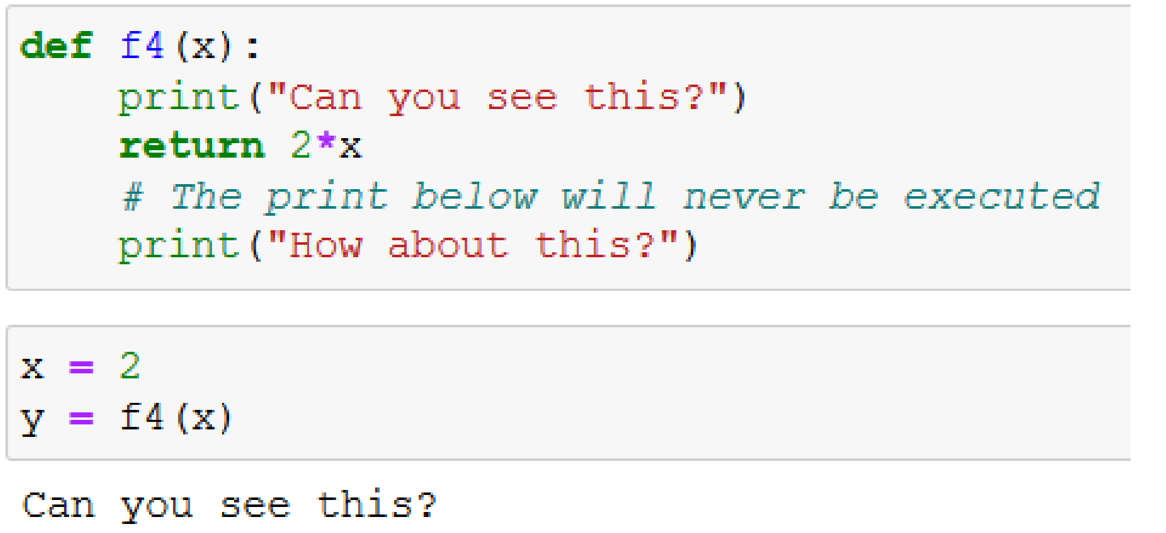
- **In-between the def and return statement (if any),** you may write lines of code to perform additional/intermediate tasks.
- **Important note:** lines of code inside the function are **indented** with 4 spaces.



```
def f4(x):
    print("Can you see this?")
    return 2*x
```

Function formatting in Python

- **In-between the def and return statement (if any),** you may write lines of code to perform additional/intermediate tasks.
- **Important note:** lines of code inside the function are **indented** with 4 spaces.
- **Important note:** once a **return** is reached and executed, the function closes and will not execute anything else.



```
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?
```

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

SCORE
0
4
2
-2
0
2
4
6
8
10
12

Activity 2: Ballistics of an angry bird

Let us practice these concepts with a second activity.

Check the notebook

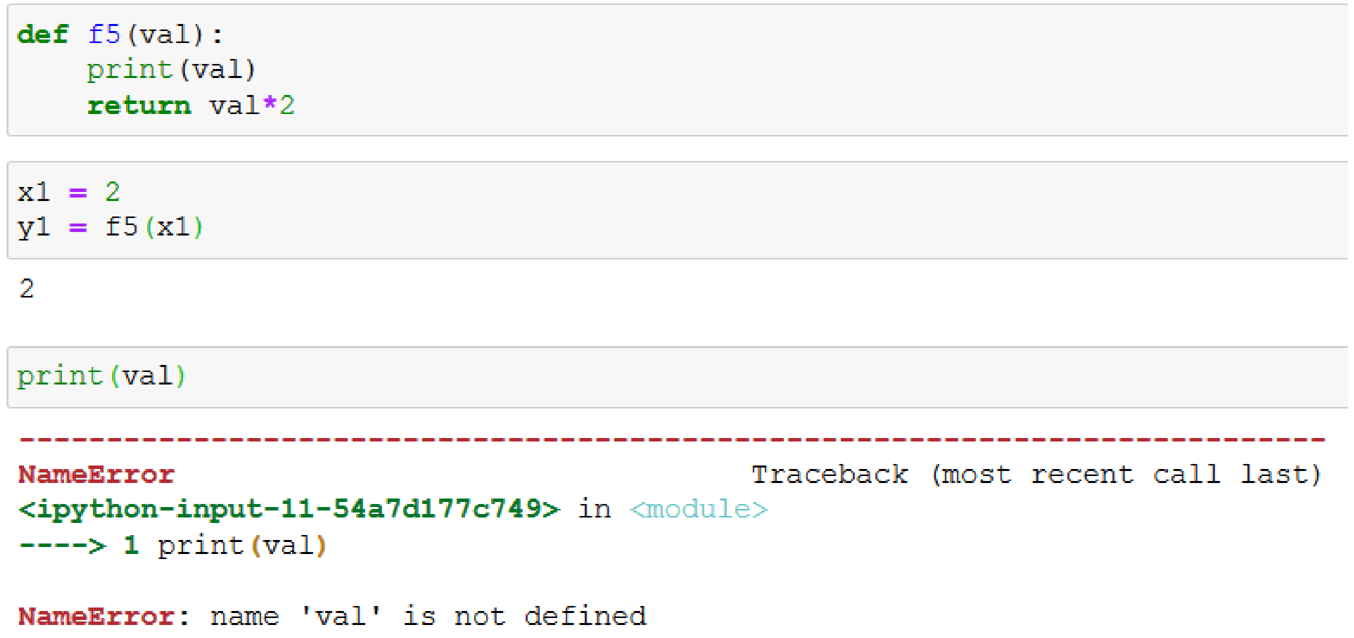
Activity 2 - Ballistics of an angry bird.ipynb

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.

Variables in functions

- **Critical importance:** variables defined **inside** the function are stored in memory while the function runs, but are cleared once the end of the function is reached. **If you need to access a variable defined inside a function, 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

```
def my_function(x):
    y = 2*x
    z = y + 3
    return z

x1 = 10
z1 = my_function(x1)
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

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)
```

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

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.

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.

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)
```

Matt’s Great advice #5

Matt’s Great Advice #5: 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.

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

Matt’s Great advice #5: example

```python
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
```

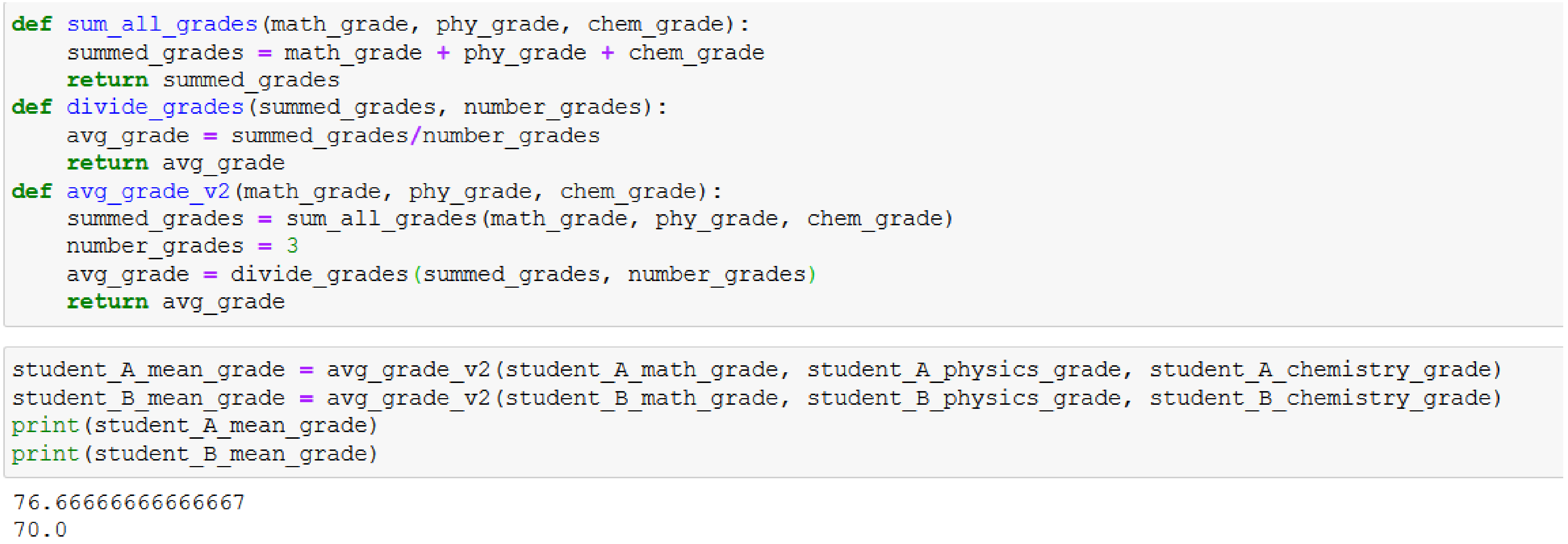
```python
student_B_math_grade = 95
student_B_physics_grade = 65
student_B_chemistry_grade = 50
student_B_sum_grade = student_B_math_grade + student_B_physics_grade + student_B_chemistry_grade
student_B_mean_grade = student_B_sum_grade/3
```

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

```python
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

- You may find useful to define several functions and call them inside other functions. Again, this makes the code more modular, professional and readable.



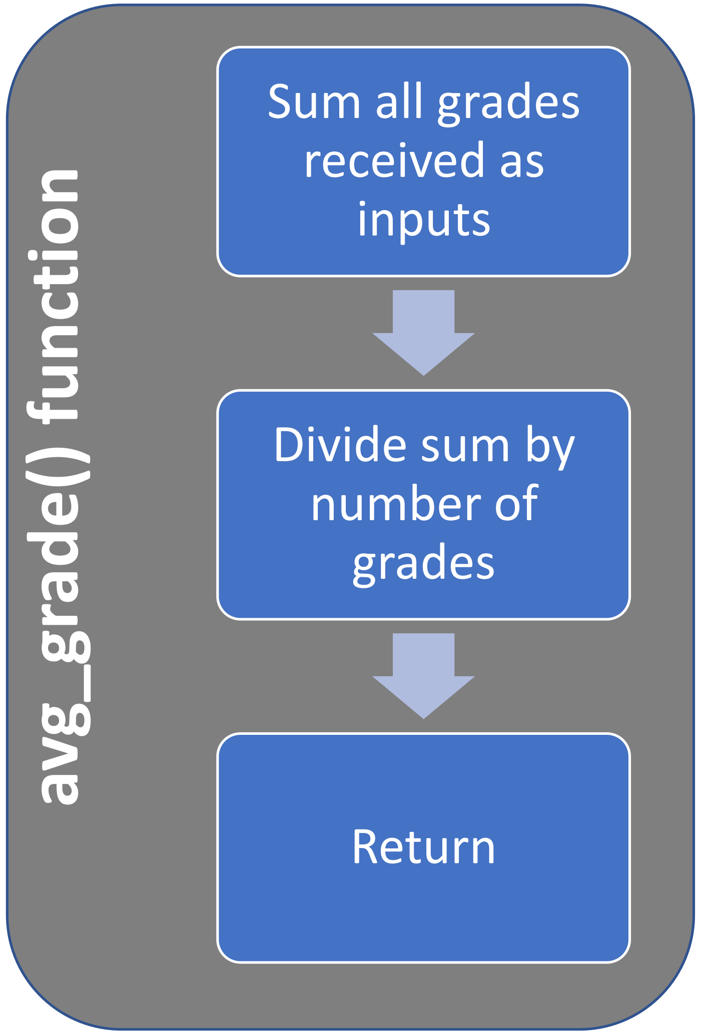
```
def sum_all_grades(math_grade, phy_grade, chem_grade):
    summed_grades = math_grade + phy_grade + chem_grade
    return summed_grades
def divide_grades(summed_grades, number_grades):
    avg_grade = summed_grades/number_grades
    return avg_grade
def avg_grade_v2(math_grade, phy_grade, chem_grade):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

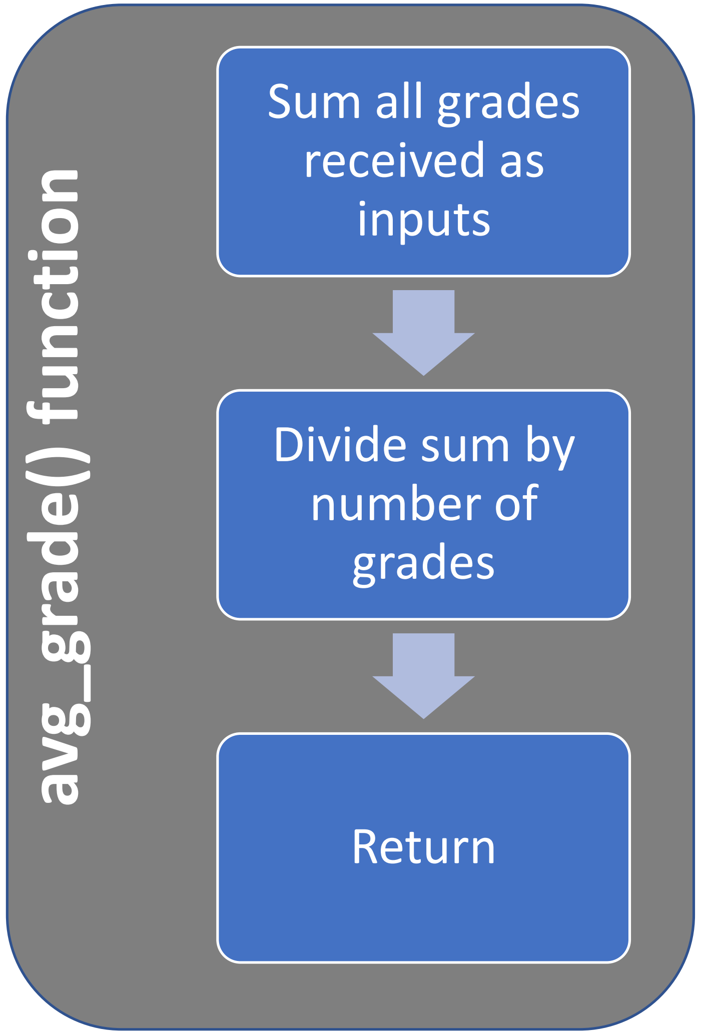
Functional diagrams

- 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.


avg_grade() function
Sum all grades received as inputs
Divide sum by number of grades
Return

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.


avg_grade() function
Sum all grades received as inputs
Divide sum by number of grades
Return

Demo (Activity 1): using a function for our guess the number game

Let me demonstrate a few additional concepts by reusing Activity 1 from earlier.

(This demo notebook is located in ./Code files/Demos, if you feel like checking it later on.)

Matt’s Great advice #6

Matt’s Great Advice #6: 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.

Conclusion

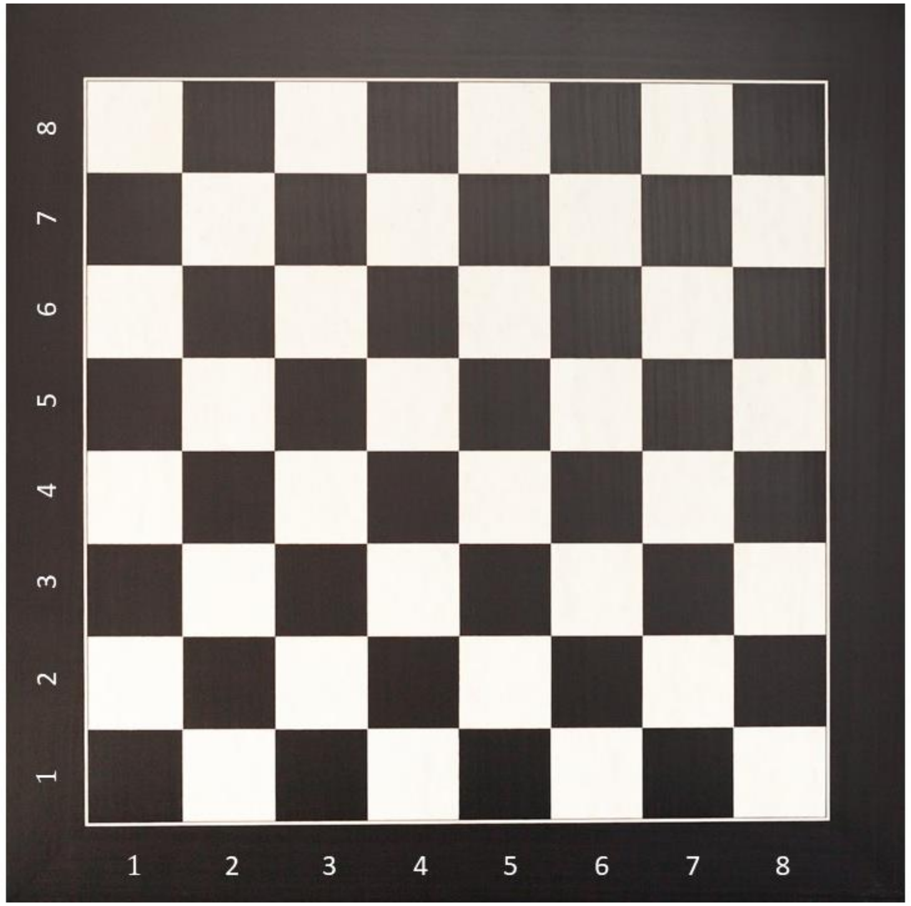
- The None type
- The Boolean type
- Boolean quiz
- Advanced concepts on Booleans
- Functions
- Memory states in functions

Activity 3: What color is the square?

Let us practice these concepts with a second activity.

Check the notebook

Activity 3 - What color is the square.ipynb



Activity 3: What color is the square?

Let us practice these concepts with a second activity.

Check the notebook

Activity 3 - What color is the square.ipynb

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

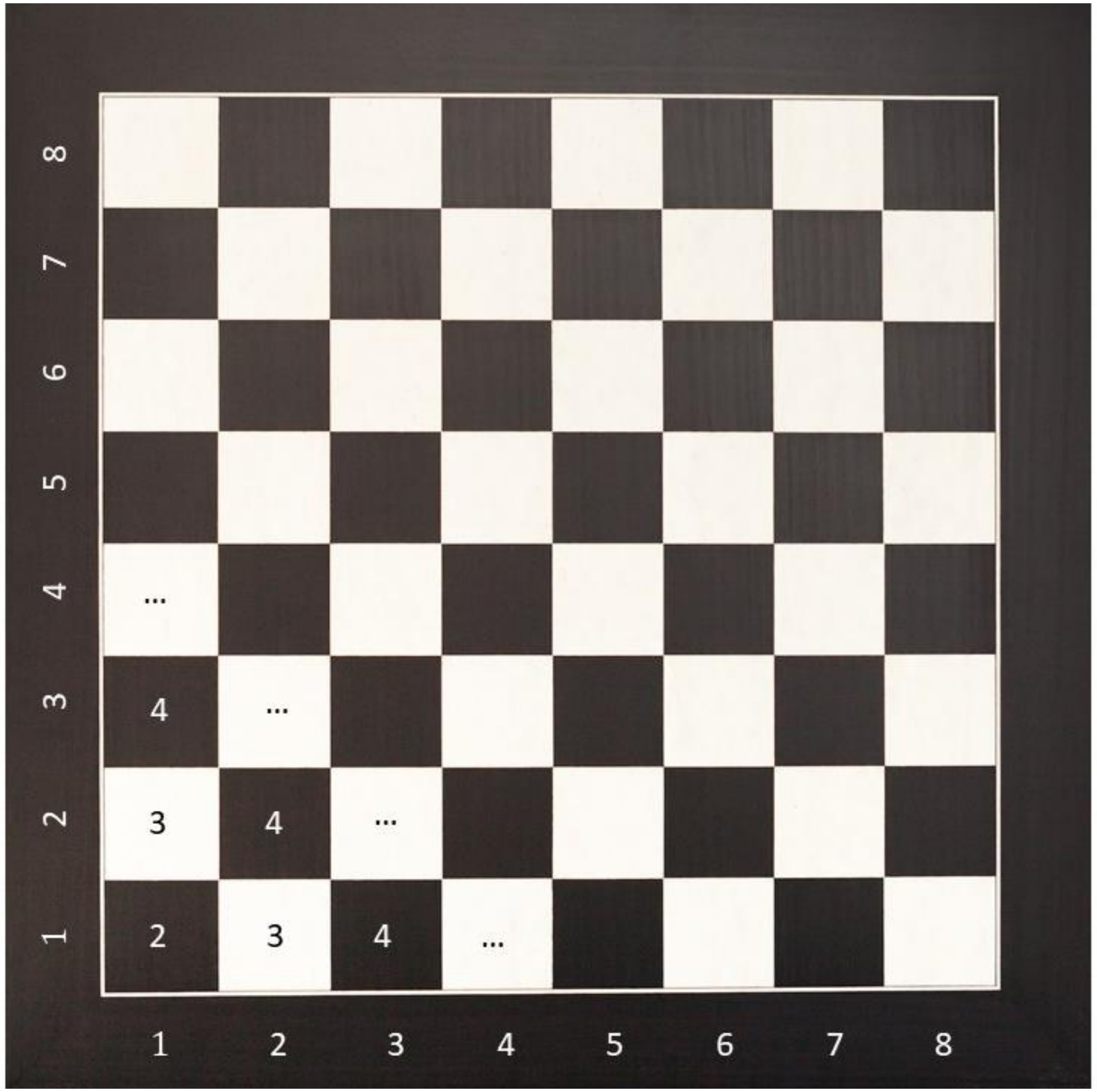
- To collect user's inputs on rows and columns indexes
- To check and print if the square is black or white.

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

Activity 3: Hint

- **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!

Activity 4: Is this triangle rectangular?

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 rectangular.
- 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.

More activities for you to practice!

Activity 5: 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 6: Distance to Point of Interest, spherical earth version

Same as Activity 5, 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 functions from the numpy library.

More activities for you to practice!

Activity 7: Guess the card game

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.