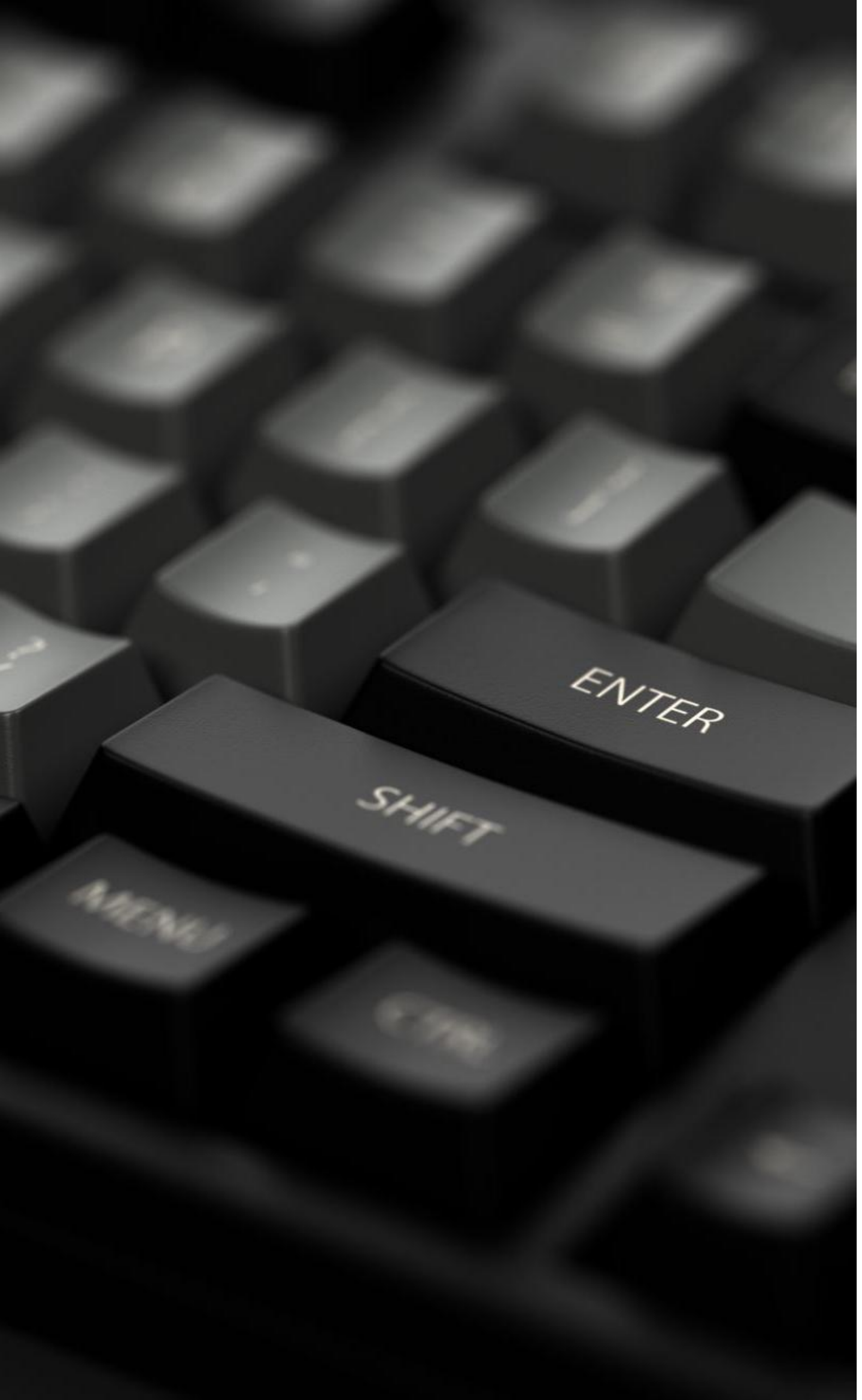
A gamified introduction to Python Programming

Lecture 2
Basic Concepts of Programming

Matthieu DE MARI – Singapore University of Technology and Design

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN



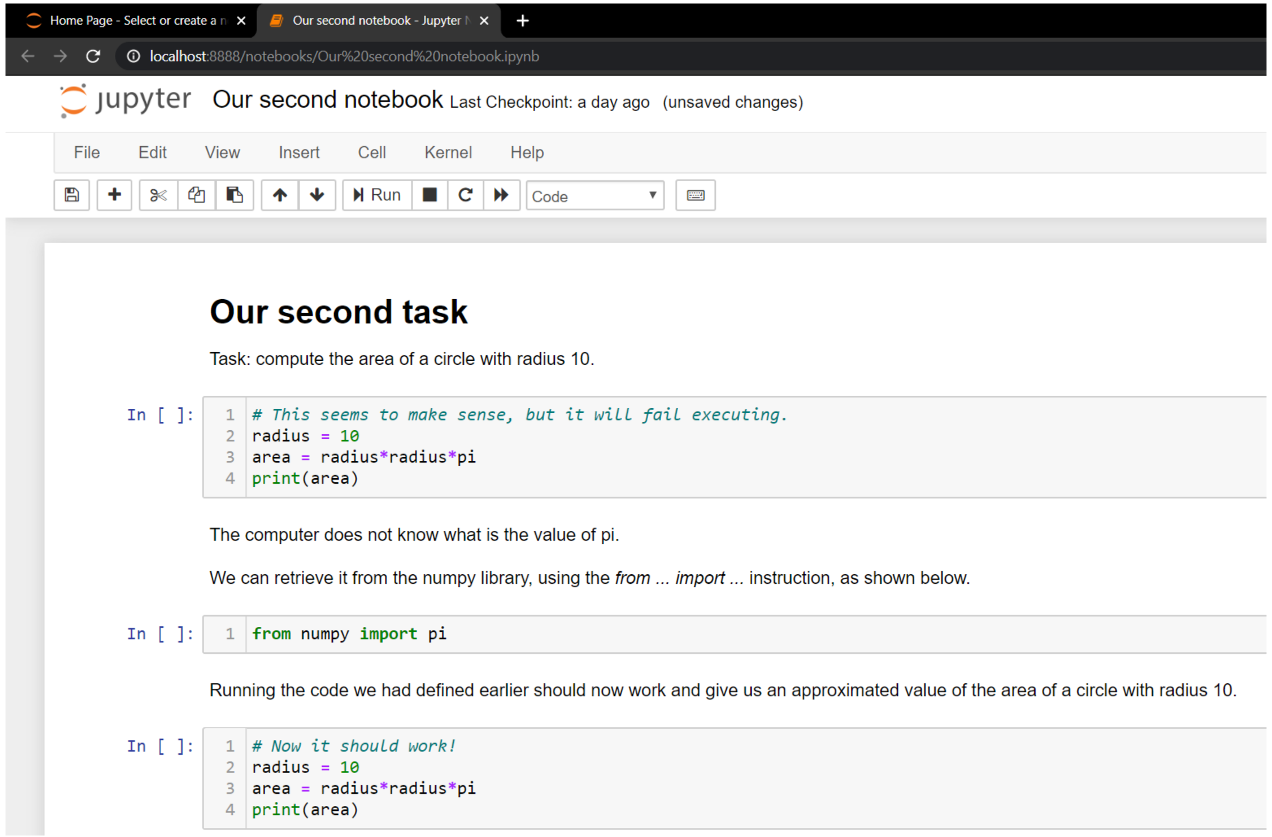
Outline (Chapter 2)

- **Everything you need to know about variables:** Naming conventions, types, IDs, assignments, types conversions.
- **Using basic math operators in Python on mathematical variables:** +, -, *, /, //, %, **.
- **Combining operators and assignments.**
- **Commenting:** in-line, block, docstring and header comments.
- **Printing and getting:** displaying with print(), getting with input().
- And more things!

Recall: our second task in Lesson 1

In this task, we have intuitively used a few concepts, such as:

1. **Assigning values to memory (variables)**
2. **Using basic math operations**
3. **Commenting your code**
4. **Printing**
5. **Executing a coding script in console or an IDE**
6. (~~Importing stuff~~, for later!)


Home Page - Select or create a n
Our second notebook - Jupyter N
localhost:8888/notebooks/Our%20second%20notebook.ipynb
jupyter
Our second notebook Last Checkpoint: a day ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Help
Run
Code

Our second task

Task: compute the area of a circle with radius 10.

```
In [ ]: # This seems to make sense, but it will fail executing.
        radius = 10
        area = radius*radius*pi
        print(area)
```

The computer does not know what is the value of pi.

We can retrieve it from the numpy library, using the *from* ... *import* ... instruction, as shown below.

```
In [ ]: from numpy import pi
```

Running the code we had defined earlier should now work and give us an approximated value of the area of a circle with radius 10.

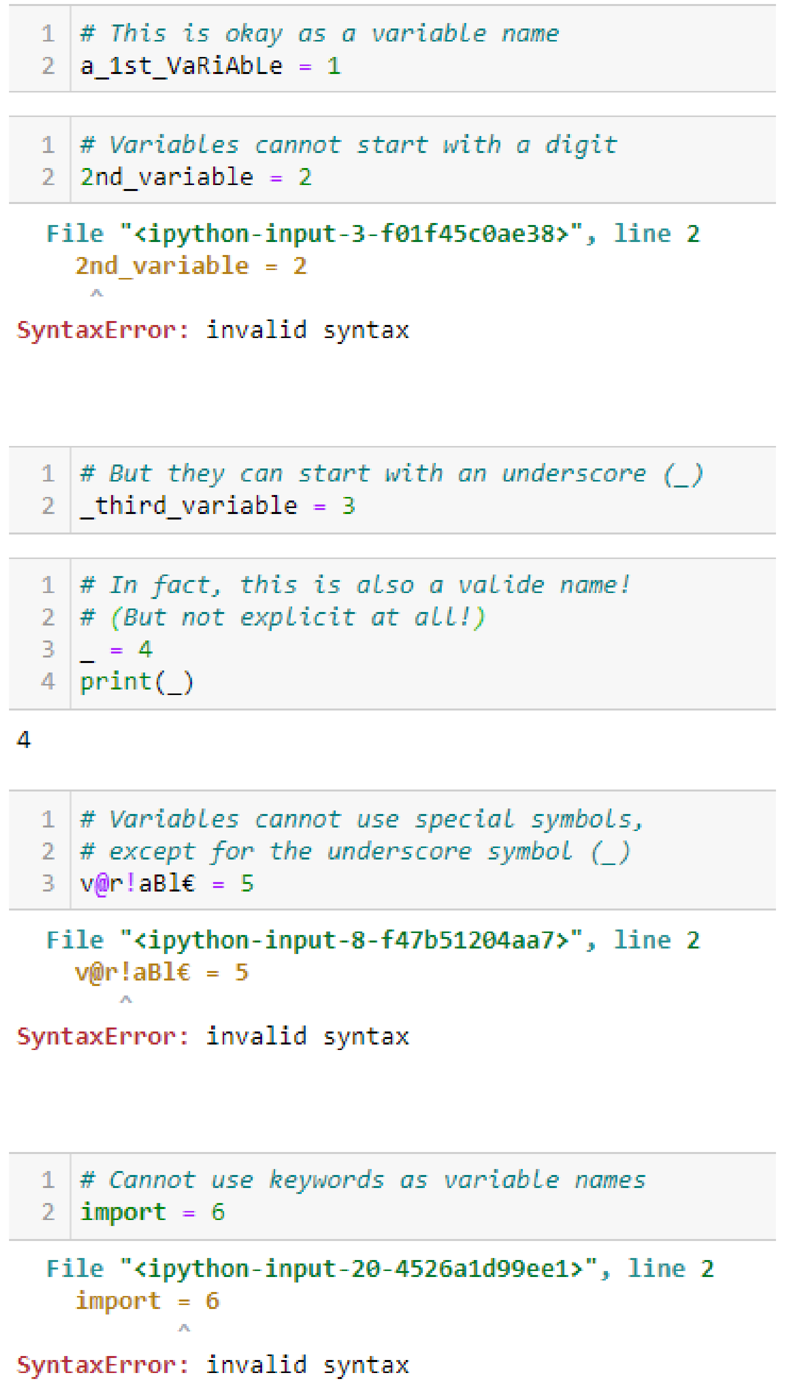
```
In [ ]: # Now it should work!
        radius = 10
        area = radius*radius*pi
        print(area)
```

→ In the next couple of slides, we will go deeper into these concepts.

About variables names

Definition (Naming Convention for variables in programming):

- **Variables names** can include any combination of **letters** (both lowercase and uppercase), **digits** and **underscore symbols** (_).
- Variables names **should NOT start with a digit**.
- Some **keywords** (e.g. import) are protected and should not be used.



```python
# This is okay as a variable name
a_1st_VaRiAbLe = 1
```

```python
# Variables cannot start with a digit
2nd_variable = 2
```

```
  File "<ipython-input-3-f01f45c0ae38>", line 2
    2nd_variable = 2
     ^
SyntaxError: invalid syntax
```

```python
# But they can start with an underscore (_)
_third_variable = 3
```

```python
# In fact, this is also a valide name!
# (But not explicit at all!)
_ = 4
print(_)
```

```
4
```

```python
# Variables cannot use special symbols,
# except for the underscore symbol (_)
v@r!aBl€ = 5
```

```
  File "<ipython-input-8-f47b51204aa7>", line 2
    v@r!aBl€ = 5
       ^
SyntaxError: invalid syntax
```

```python
# Cannot use keywords as variable names
import = 6
```

```
  File "<ipython-input-20-4526a1d99ee1>", line 2
    import = 6
           ^
SyntaxError: invalid syntax
```

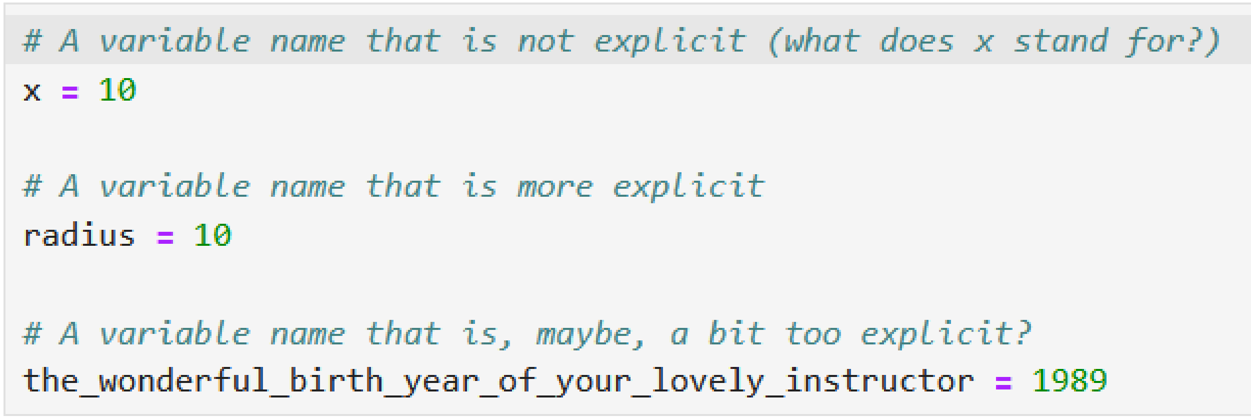
Matt’s Great Advice

Matt’s Great Advice: Try making your variables names explicit.

It is a good idea to make your **variables names** explicit (but not too much!) rather than using meaningless ones that leave your reader guessing what they stand for (x, y, a, b1, c2, etc.).

You can **use underscores** if you find it helpful (e.g. current_year = 2020).

Your future self, and colleagues, will thank you, when they work on your code later on.



```
# A variable name that is not explicit (what does x stand for?)
x = 10

# A variable name that is more explicit
radius = 10

# A variable name that is, maybe, a bit too explicit?
the_wonderful_birth_year_of_your_lovely_instructor = 1989
```

slido

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

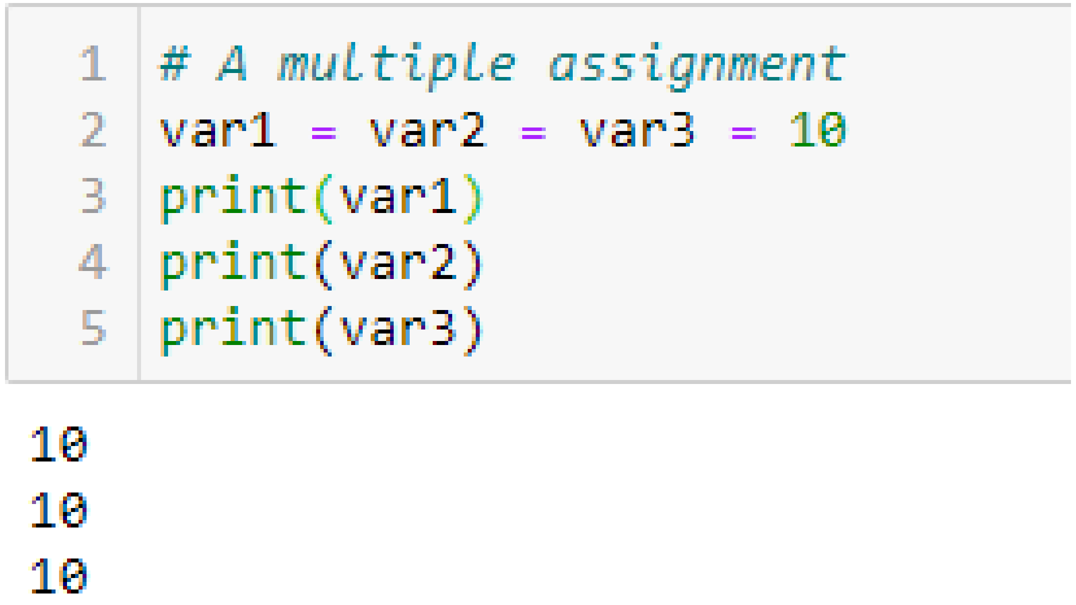
What happens if you use the command "print = 5" in Python?

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

Multiple assignments

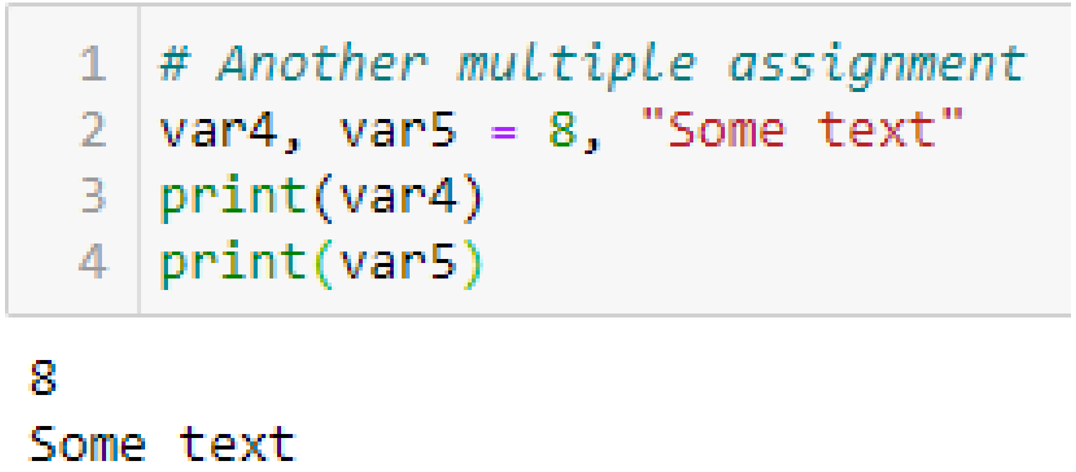
Python also allows **multiple assignments** using

1. **Successive multiple equal signs:** the value on the far right is assigned to all variables names identically.
2. **Or by having several variables names and values separated by commas on both sides:** the values are respectively assigned to the variables names, in a one for one manner.



```python
# A multiple assignment
var1 = var2 = var3 = 10
print(var1)
print(var2)
print(var3)
```

```
10
10
10
```



```python
# Another multiple assignment
var4, var5 = 8, "Some text"
print(var4)
print(var5)
```

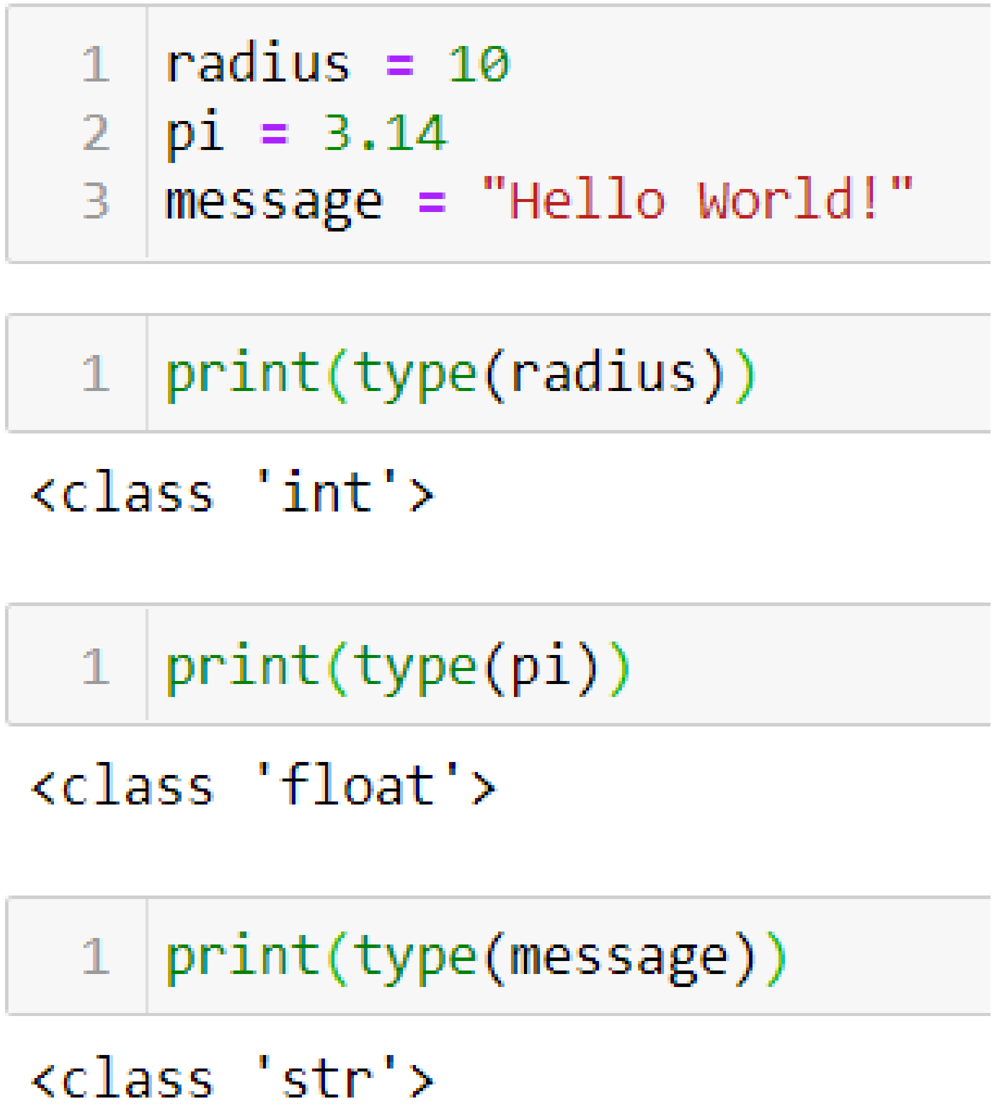
```
8
Some text
```

Variable types

Definition (variable type/class):
Everything in your computer is stored in memory as 0s and 1s.
A **variable type/class** defines how the sequence of 0s and 1s should be interpreted by the computer.

Let us start with three basic types:

- **int:** an integer number.
- **float:** a floating point number, for decimal numbers.
- **str:** stands for string, a bunch of text.



```python
radius = 10
pi = 3.14
message = "Hello World!"
```

```python
print(type(radius))
```

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

```python
print(type(pi))
```

```
<class 'float'>
```

```python
print(type(message))
```

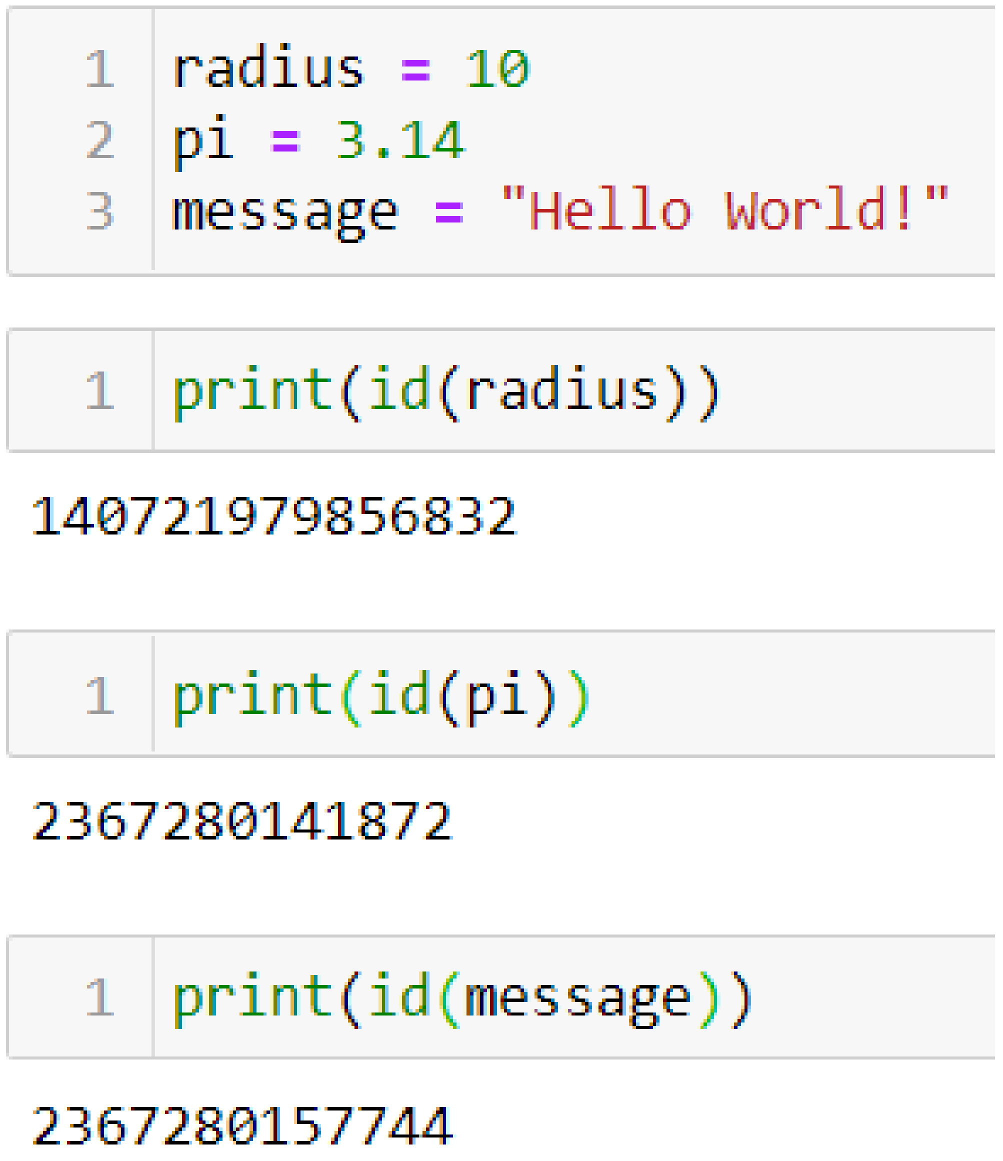
```
<class 'str'>
```

Variable IDs

Definition (variable ID):
Everything in your computer is stored in memory as 0s and 1s.

A **variable ID** returns a number, corresponding to a memory address where the variable is being stored in memory.

The storing process for variables is automatically handled by your computer. More on this later, so let us keep it in mind.



```python
radius = 10
pi = 3.14
message = "Hello World!"
```

```python
print(id(radius))
```

```
140721979856832
```

```python
print(id(pi))
```

```
2367280141872
```

```python
print(id(message))
```

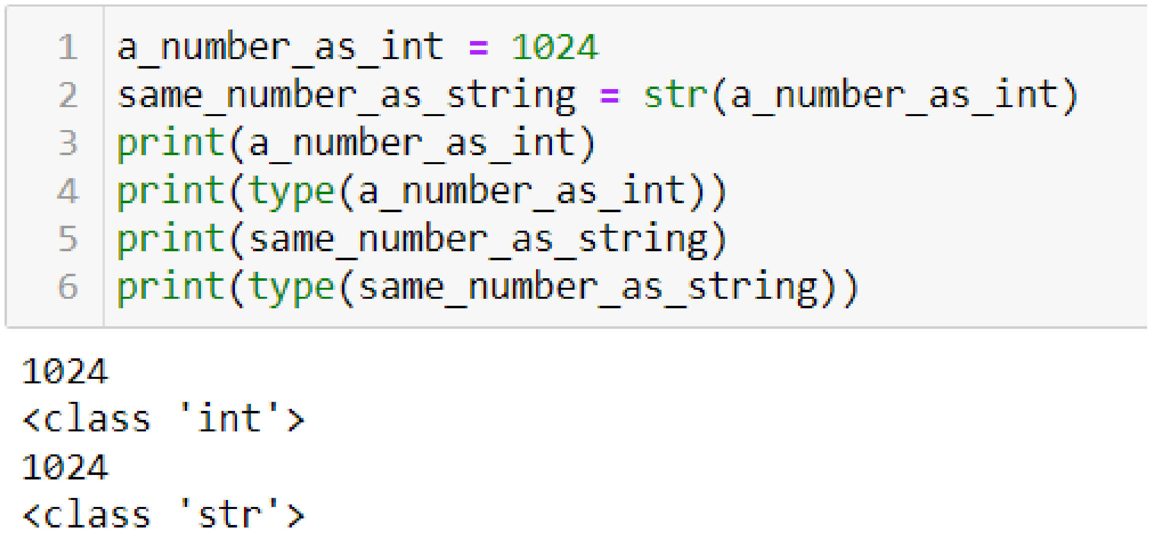
```
2367280157744
```

Type conversion

Int/Float → String: You can convert any int/float to a string using the **str()** function.

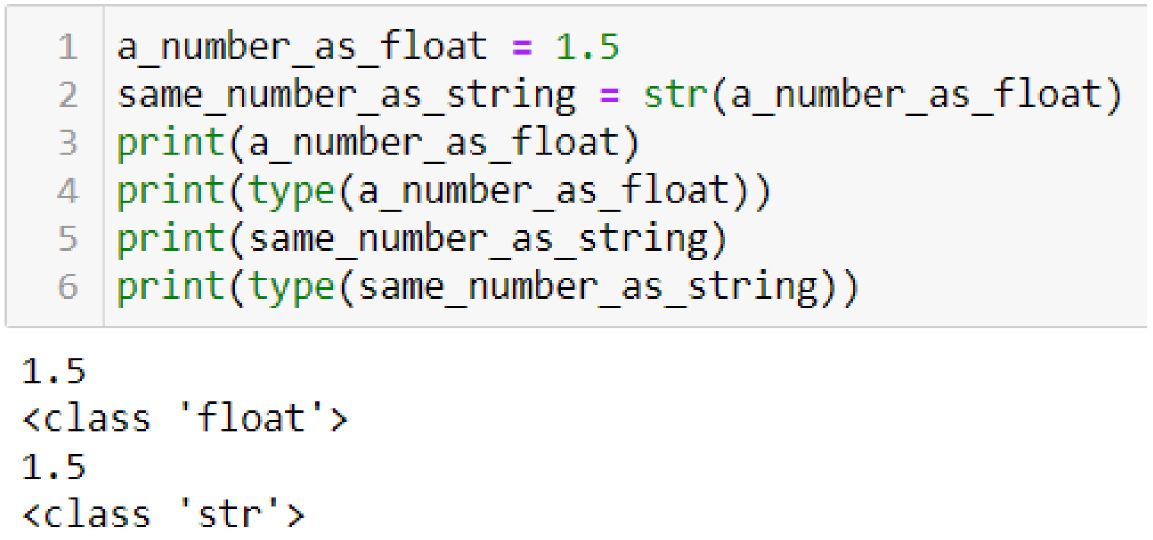
- **Note:** Both the int/float and the str variables might print the same thing on screen, but they are not equivalent when it comes to what is stored in memory and how these variables can be used!

  For your computer, these two variables are very different things!



```python
a_number_as_int = 1024
same_number_as_string = str(a_number_as_int)
print(a_number_as_int)
print(type(a_number_as_int))
print(same_number_as_string)
print(type(same_number_as_string))
```

```
1024
<class 'int'>
1024
<class 'str'>
```



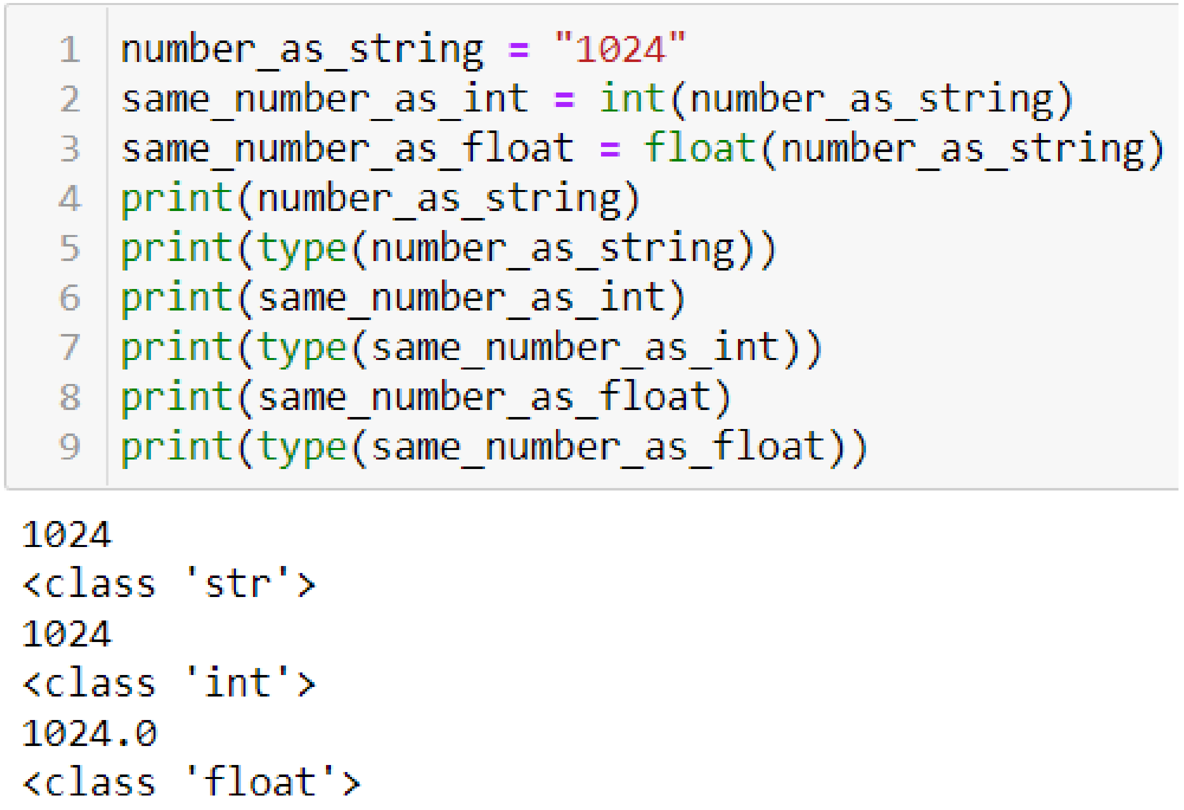
```python
a_number_as_float = 1.5
same_number_as_string = str(a_number_as_float)
print(a_number_as_float)
print(type(a_number_as_float))
print(same_number_as_string)
print(type(same_number_as_string))
```

```
1.5
<class 'float'>
1.5
<class 'str'>
```

Type conversion

String → Int/Float: You can convert a string of digits to an int/float using **int()** or **float().**

- A string containing only digits can be converted to int or float.
- A string of digits may contain a single decimal point. In that case, it can be converted to a float, but not to an int.
- Everything else fails (two decimal points, special characters, etc.).



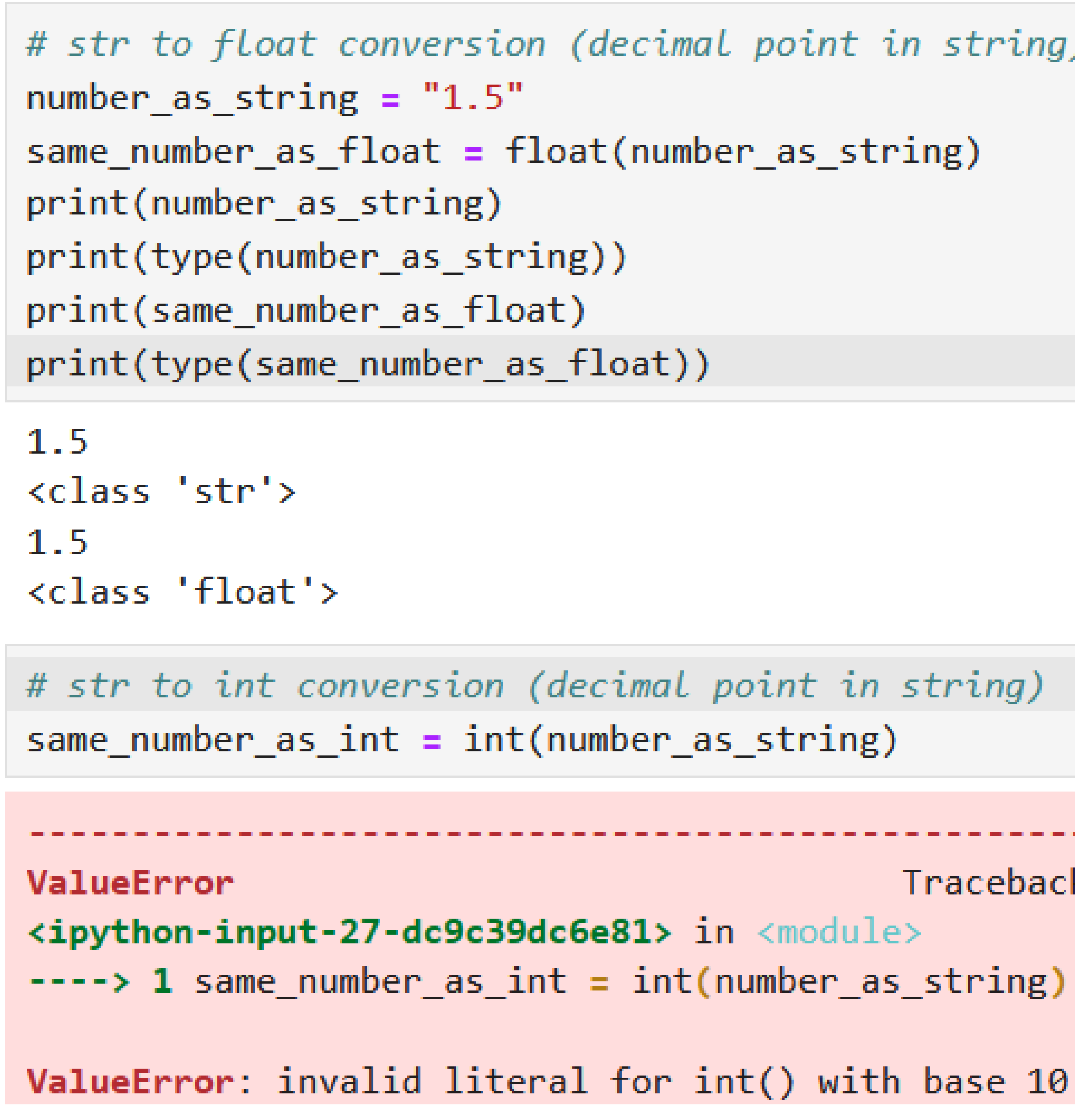
```python
number_as_string = "1024"
same_number_as_int = int(number_as_string)
same_number_as_float = float(number_as_string)
print(number_as_string)
print(type(number_as_string))
print(same_number_as_int)
print(type(same_number_as_int))
print(same_number_as_float)
print(type(same_number_as_float))
```

```
1024
<class 'str'>
1024
<class 'int'>
1024.0
<class 'float'>
```

Type conversion

String → Int/Float: You can convert a string of digits to an int/float using **int()** or **float().**

- A string containing only digits can be converted to int or float.
- A string of digits may contain a single decimal point. In that case, it can be converted to a float, but not to an int.
- Everything else fails (two decimal points, special characters, etc.).



```
# str to float conversion (decimal point in string
number_as_string = "1.5"
same_number_as_float = float(number_as_string)
print(number_as_string)
print(type(number_as_string))
print(same_number_as_float)
print(type(same_number_as_float))
```

```
1.5
<class 'str'>
1.5
<class 'float'>
```

```
# str to int conversion (decimal point in string)
same_number_as_int = int(number_as_string)
```

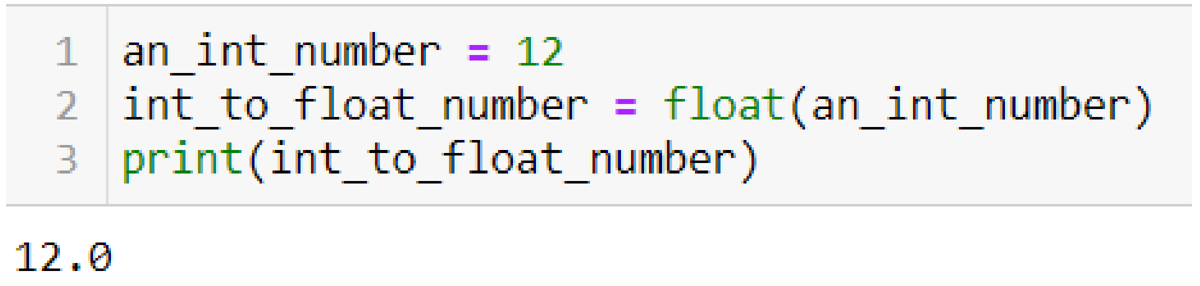
```
---------------------------------------------------------
ValueError                                      Tracebac
<ipython-input-27-dc9c39dc6e81> in <module>
----> 1 same_number_as_int = int(number_as_string)

ValueError: invalid literal for int() with base 10
```

Type conversion

Int ←→ Float: You can convert a float to an int and vice versa.

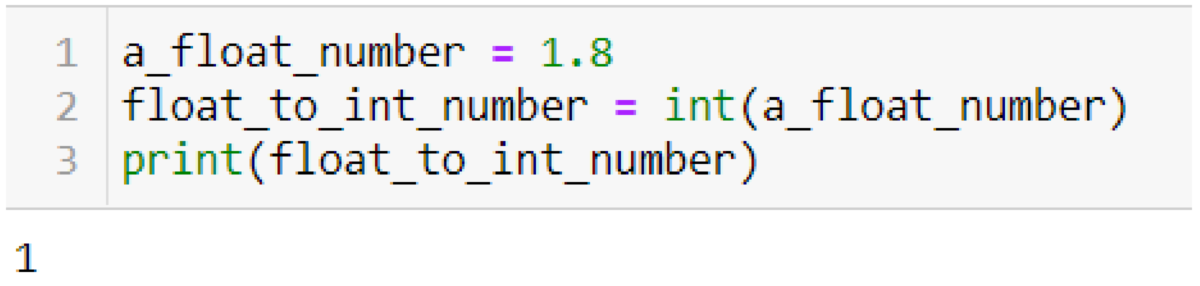
- Converting from int to float adds a single 0 decimal.
- Also, converting from float to int, will **drop all decimals** (i.e. a truncation of all decimals, which is NOT the same as rounding!).

Feel free to experiment!



```python
an_int_number = 12
int_to_float_number = float(an_int_number)
print(int_to_float_number)
```

```
12.0
```



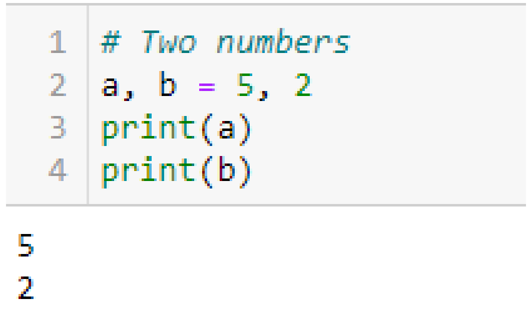
```python
a_float_number = 1.8
float_to_int_number = int(a_float_number)
print(float_to_int_number)
```

```
1
```

Python basic operators

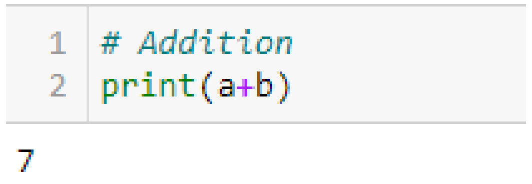
By default, Python comes with a few **basic math operators, for number (int/float) variables**.

- **+:** addition
- **-:** subtraction
- ***:** multiplication
- **/:** division
- **//:** integer division
- **%:** remaining of the integer division (a.k.a. modulus)
- ****:** exponentiation



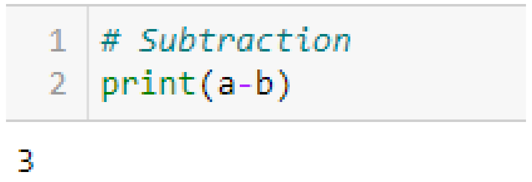
```
# Two numbers
a, b = 5, 2
print(a)
print(b)
```

```
5
2
```



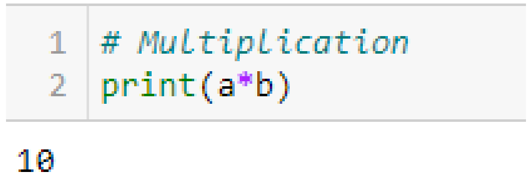
```
# Addition
print(a+b)
```

```
7
```



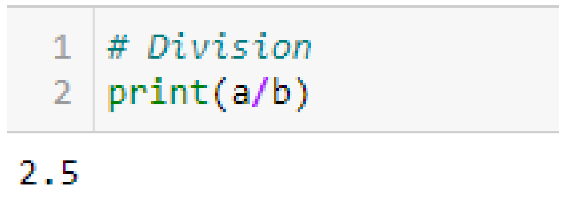
```
# Subtraction
print(a-b)
```

```
3
```



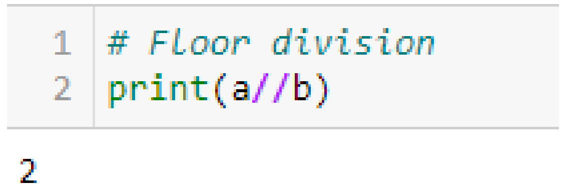
```
# Multiplication
print(a*b)
```

```
10
```



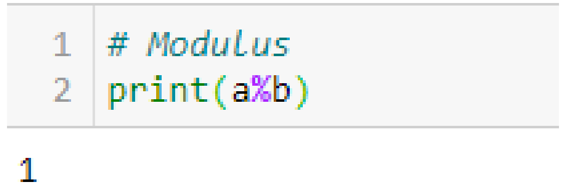
```
# Division
print(a/b)
```

```
2.5
```



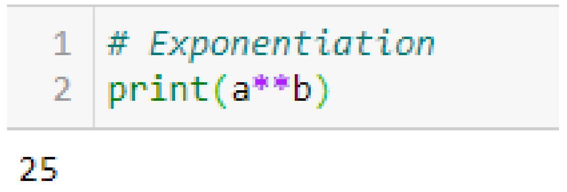
```
# Floor division
print(a//b)
```

```
2
```



```
# Modulus
print(a%b)
```

```
1
```



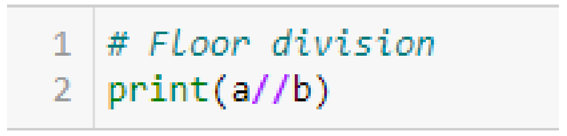
```
# Exponentiation
print(a**b)
```

```
25
```

Python basic operators

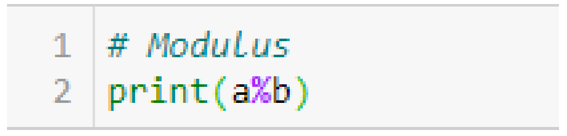
By default, Python comes with a few **basic math operators, for number (int/float) variables**.

- **+:** addition
- **-:** subtraction
- ***:** multiplication
- **/:** division
- **//: integer division**
- **%: remaining of the integer division (a.k.a. modulus)**
- ****:** exponentiation



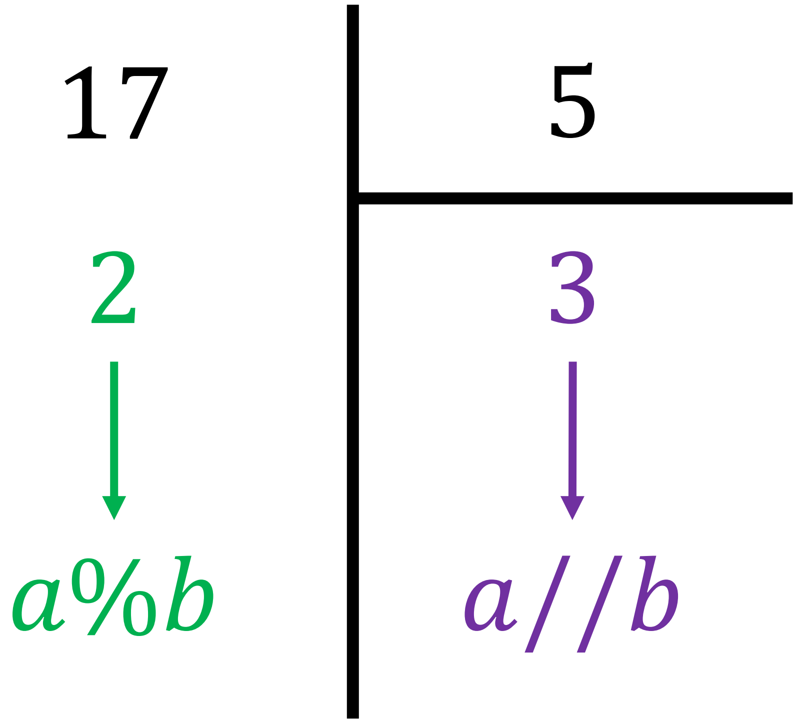
```
# Floor division
print(a//b)
```

```
2
```



```
# Modulus
print(a%b)
```

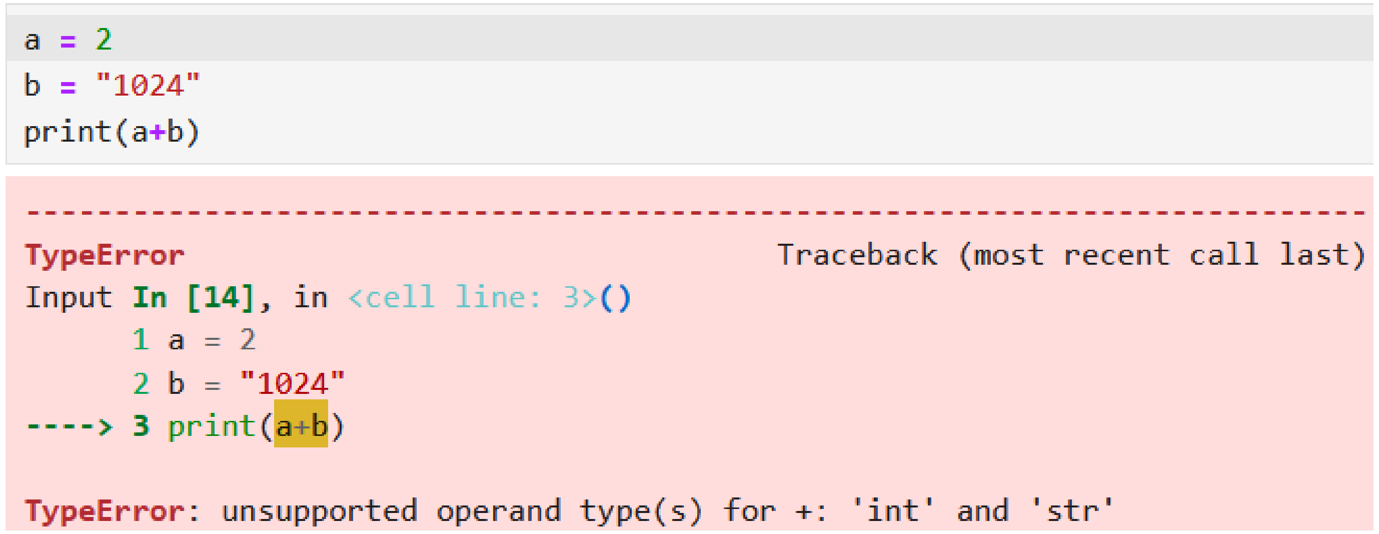
```
1
```



Python basic operators

By default, Python comes with a few **basic math operators,** **for number (int/float) variables**.

- **+:** addition
- **-:** subtraction
- ***:** multiplication
- **/:** division
- **//:** integer division
- **%:** remaining of the integer division (a.k.a. modulus)
- ****:** exponentiation



```
a = 2
b = "1024"
print(a+b)
```

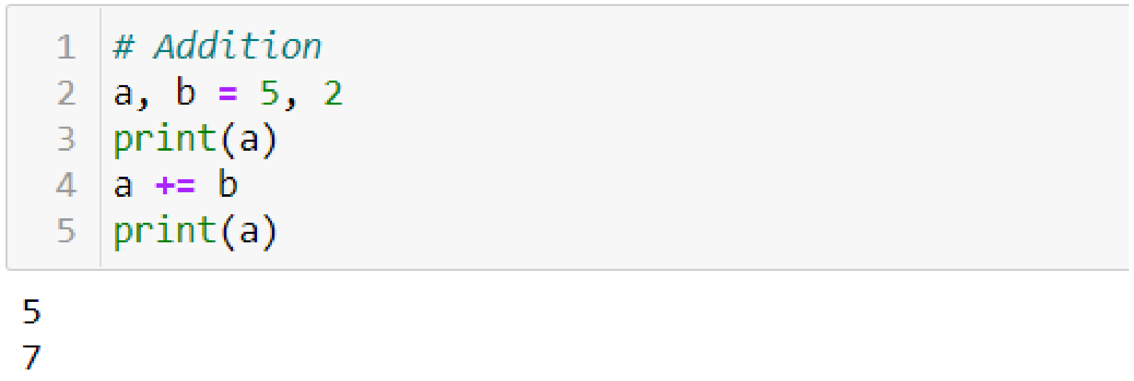
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [14], in <cell line: 3>()
      1 a = 2
      2 b = "1024"
----> 3 print(a+b)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Combining operators and assignments

You can mix both mathematical and memory assignments operator.

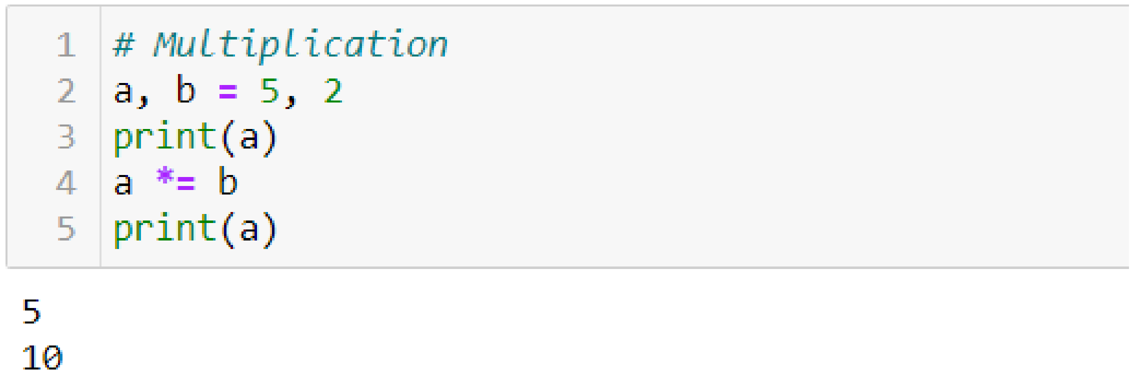
- For instance, the **operation +=** sums values on the right- and left-hand sides and assigns the result to the variable on the left-hand side.

- Similarly, we have the following operators: **-=, *=, /=, //=, %=, **=.**
- Can you guess what they do?



```python
# Addition
a, b = 5, 2
print(a)
a += b
print(a)
```

```
5
7
```



```python
# Multiplication
a, b = 5, 2
print(a)
a *= b
print(a)
```

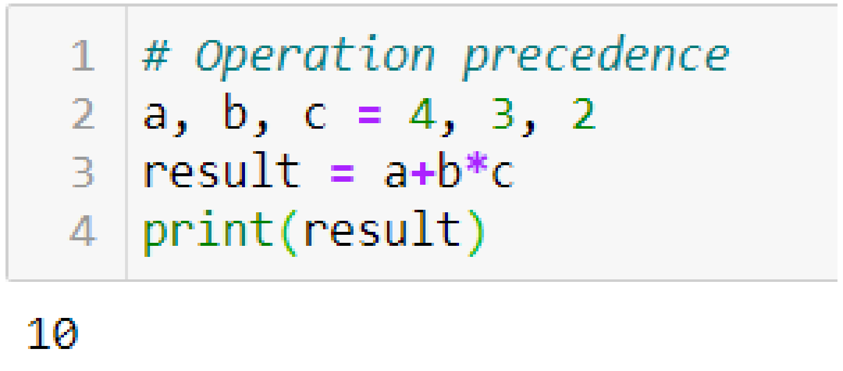
```
5
10
```

Operation precedence

Precedence order in Python: Operations are always computed **from left to right**.

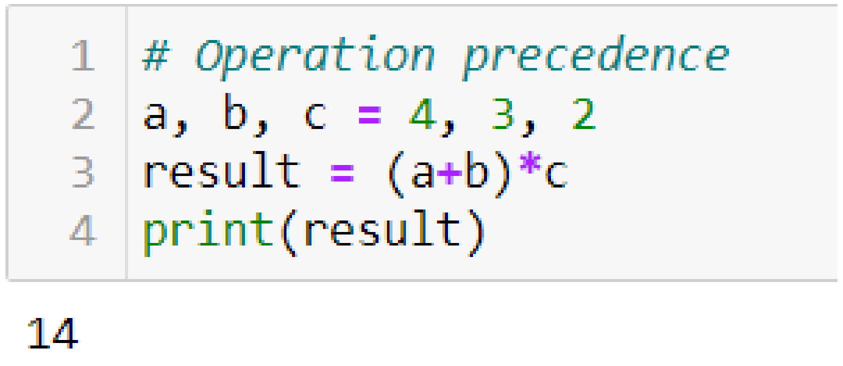
However, **operation precedence rules** apply:

- **Exponentiation** has higher precedence than **multiplications/divisions/modulus**, which have higher precedence than **additions/subtractions**.
- As in mathematics, **parentheses** can be used to “force” precedence, as they will always have the highest precedence.



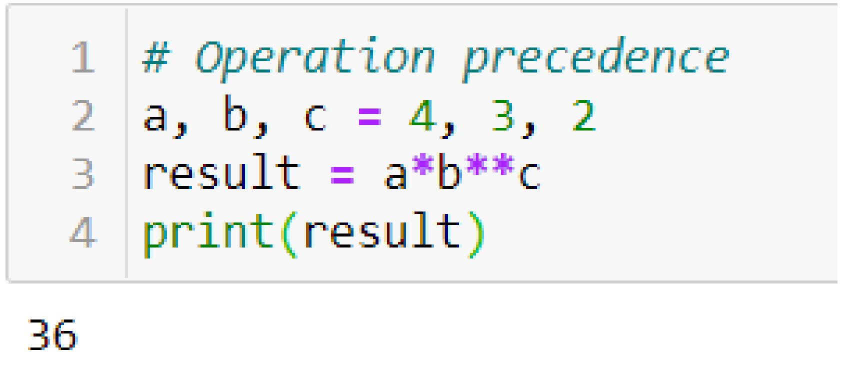
```
# Operation precedence
a, b, c = 4, 3, 2
result = a+b*c
print(result)
```

```
10
```



```
# Operation precedence
a, b, c = 4, 3, 2
result = (a+b)*c
print(result)
```

```
14
```



```
# Operation precedence
a, b, c = 4, 3, 2
result = a*b**c
print(result)
```

```
36
```

Activity 1: Compute the roots of a quadratic equation, with strictly positive determinant

Consider the quadratic equation

$$ax^2 + bx + c = 0$$

With $a, b, c \in \mathbb{R}$, such that the determinant $\Delta > 0$

$$\Delta = b^2 - 4ac$$

As seen in math classes, the two roots of this equation (x_1, x_2) are given by

$$x_1 = \frac{-b + \sqrt{\Delta}}{2a} \quad and \quad x_2 = \frac{-b - \sqrt{\Delta}}{2a}$$

- **Task:** Consider a quadratic equation with the following values: $\boldsymbol{a = 2}$, $\boldsymbol{b = -2}$ and $\boldsymbol{c = -24}$.

Write a Python program that

1. computes the value of the **determinant** Δ and **prints** it on screen (to check it is strictly positive),
2. And, later on, computes the values of the **roots** x_1 and x_2 and **prints** them on screen.

Activity 1: Compute the roots of a quadratic equation, with strictly positive determinant

- For this activity, you may now open and use the notebook

  Activity 1 - computing the roots of a quadratic equation.ipynb

- Remember to start a notebook environment using the command below, in a console, with location appropriately selected

  py –m notebook

- Also, remember, the **square root of x** is equivalent to x **being exponentiated to the power 0.5 (or import numpy sqrt function!)**

$$\sqrt{x} = x^{0.5} = x ** (0.5)$$

Activity 1: Compute the roots of a quadratic equation, with strictly positive determinant

Consider the quadratic equation

$$ax^2 + bx + c = 0$$

With $a, b, c \in \mathbb{R}$, such that the determinant $\Delta > 0$

$$\Delta = b^2 - 4ac$$

As seen in math classes, the two roots of this equation (x_1, x_2) are given by

$$x_1 = \frac{-b + \sqrt{\Delta}}{2a} \quad and \quad x_2 = \frac{-b - \sqrt{\Delta}}{2a}$$

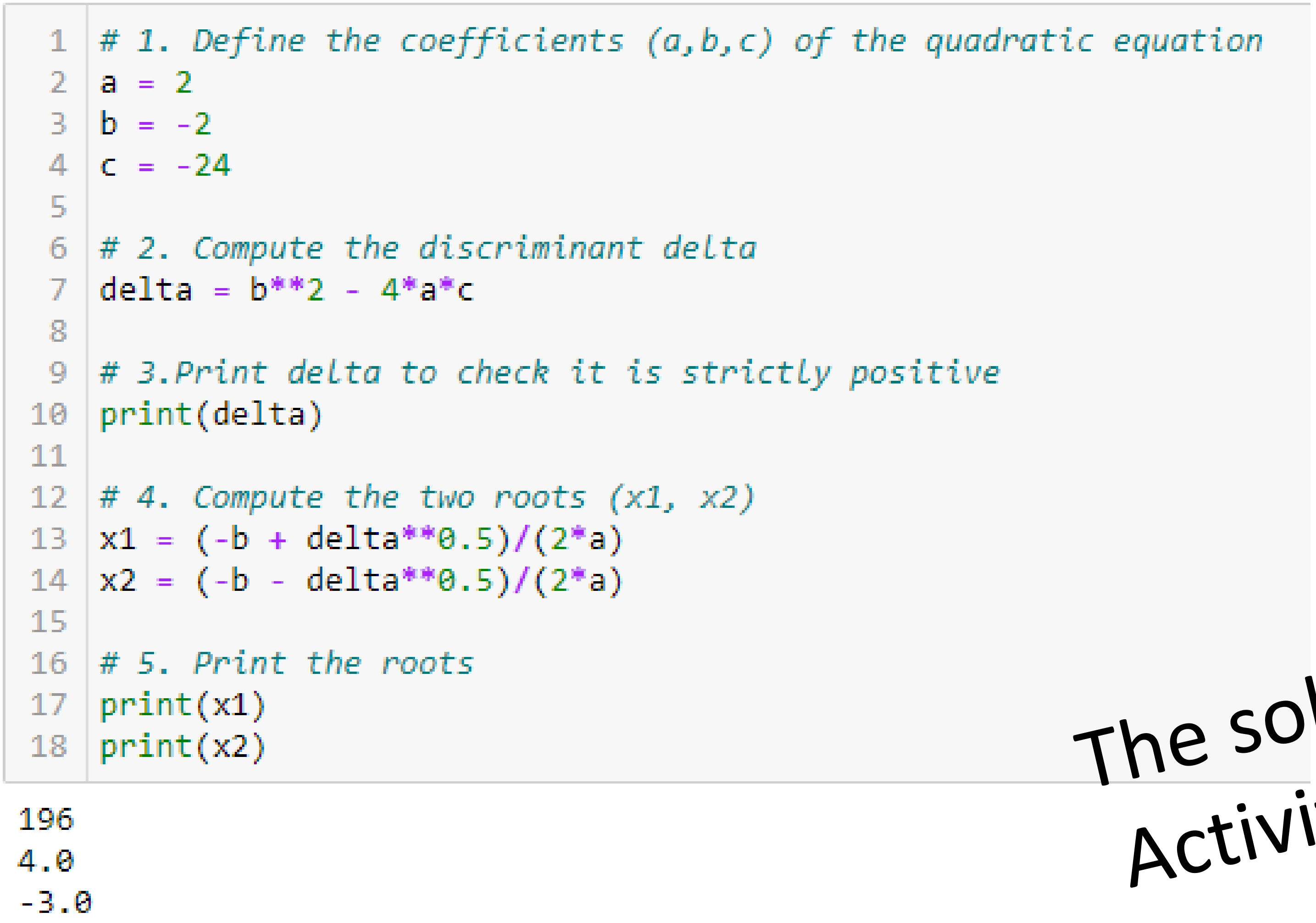
- **Task:** Consider a quadratic equation with the following values: $\boldsymbol{a = 2}$, $\boldsymbol{b = -2}$ and $\boldsymbol{c = -24}$.

Write a Python program that

1. computes the value of the **determinant** Δ and **prints** it on screen (to check it is strictly positive),
2. And, later on, computes the values of the **roots** x_1 and x_2 and **prints** them on screen.

→ **Expected answers: $\boldsymbol{\Delta = 196, x_1 = 4, x_2 = -3}$.**

Activity 1: Solution



```
# 1. Define the coefficients (a,b,c) of the quadratic equation
a = 2
b = -2
c = -24

# 2. Compute the discriminant delta
delta = b**2 - 4*a*c

# 3.Print delta to check it is strictly positive
print(delta)

# 4. Compute the two roots (x1, x2)
x1 = (-b + delta**0.5)/(2*a)
x2 = (-b - delta**0.5)/(2*a)

# 5. Print the roots
print(x1)
print(x2)
```

```
196
4.0
-3.0
```

The solved notebook is in the Activity Solutions subfolder!

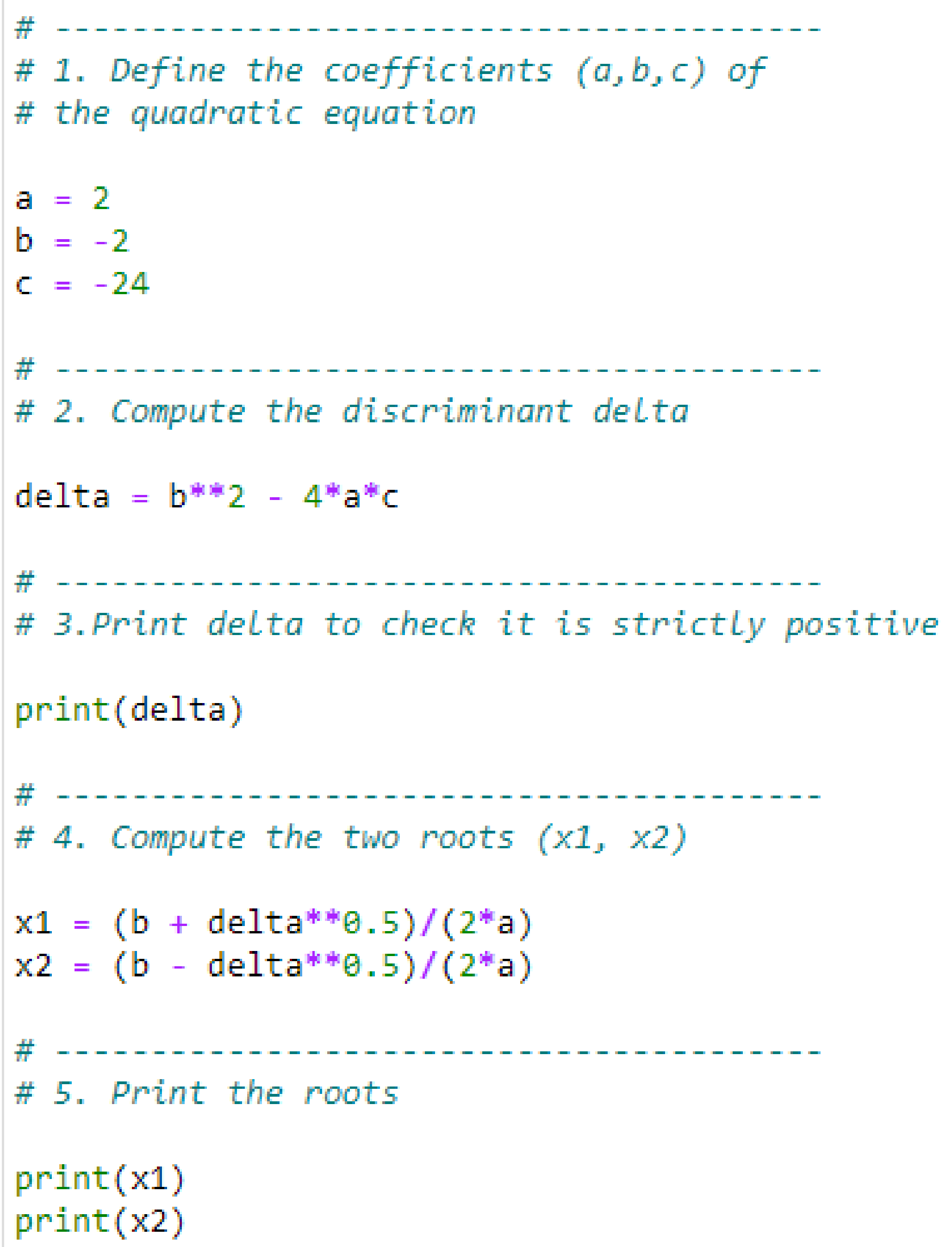
Single line comments

Definition (single line comments):

Single line comments help the reader understand what is happening in your code.

On heavy programming projects, with several programmers, these are **simply essential**.

Single line comments are also great if you want to use **separators** in your code. Personally, I use **# ---------------**.



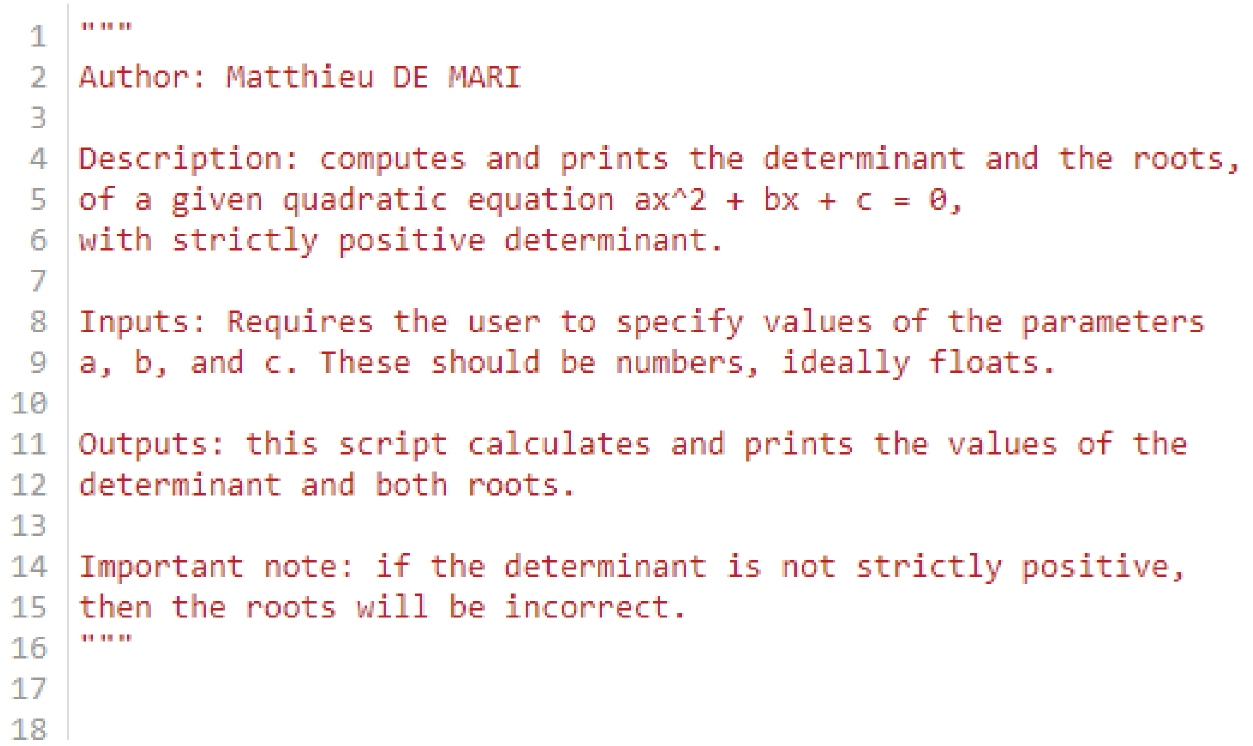
```python
# ----------------------------------------
# 1. Define the coefficients (a,b,c) of
# the quadratic equation

a = 2
b = -2
c = -24

# ----------------------------------------
# 2. Compute the discriminant delta

delta = b**2 - 4*a*c

# ----------------------------------------
# 3.Print delta to check it is strictly positive

print(delta)

# ----------------------------------------
# 4. Compute the two roots (x1, x2)

x1 = (b + delta**0.5)/(2*a)
x2 = (b - delta**0.5)/(2*a)

# ----------------------------------------
# 5. Print the roots

print(x1)
print(x2)
```

Block comments

Definition (block comments):

Block comments are comments starting and ending with **triple quotation marks (“”” or ‘’’)**.

These are great for **headers** in your files. Typically used, for instance, to inform the reader about the author, give a brief summary of what the code does, etc.



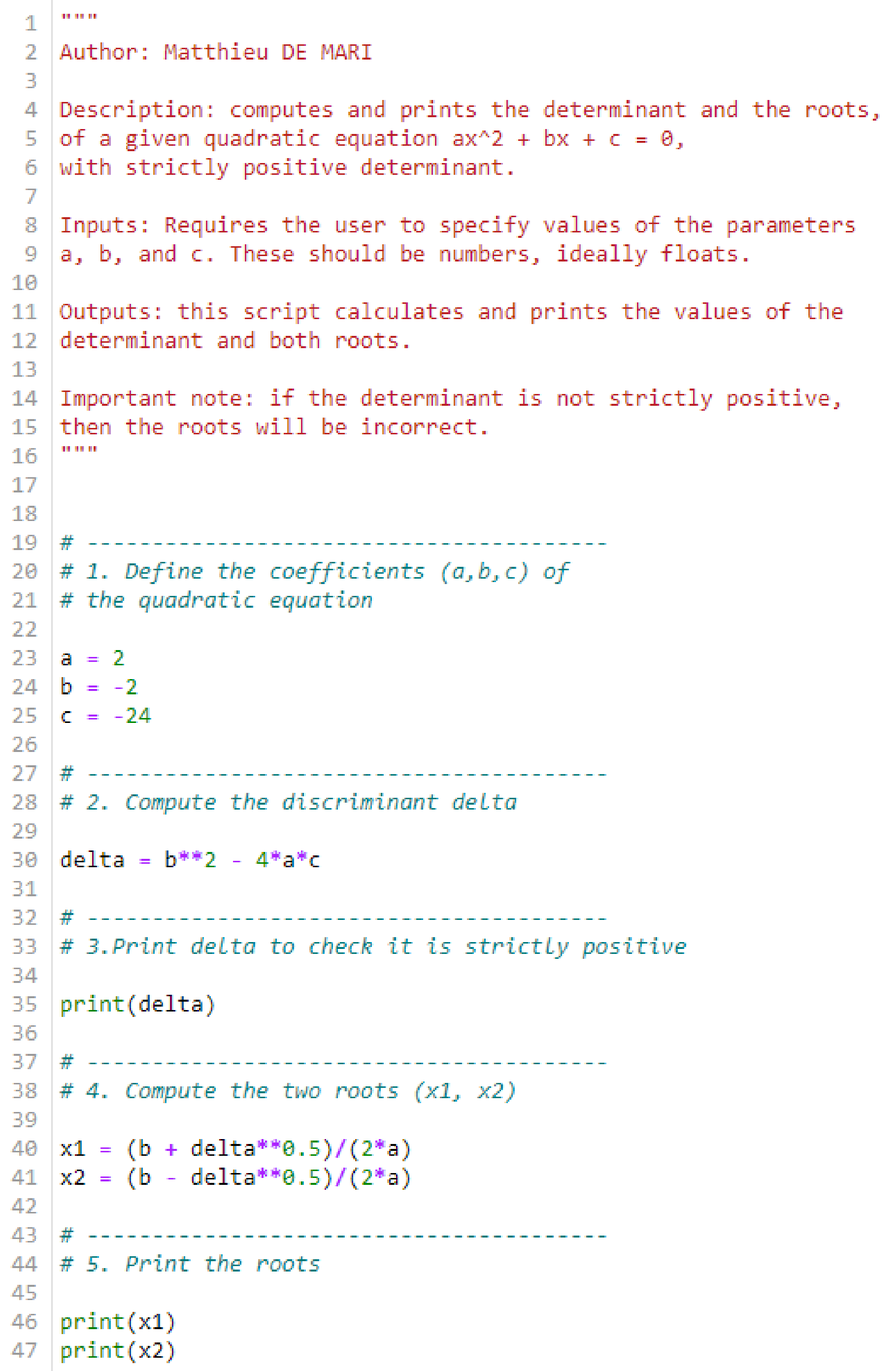
```
"""
Author: Matthieu DE MARI

Description: computes and prints the determinant and the roots,
of a given quadratic equation ax^2 + bx + c = 0,
with strictly positive determinant.

Inputs: Requires the user to specify values of the parameters
a, b, and c. These should be numbers, ideally floats.

Outputs: this script calculates and prints the values of the
determinant and both roots.

Important note: if the determinant is not strictly positive,
then the roots will be incorrect.
"""


```

Block comments

Definition (block comments):

Block comments are comments starting and ending with **triple quotation marks (“”” or ‘’’)**.

These are great for **headers** in your files. Typically used, for instance, to inform the reader about the author, give a brief summary of what the code does, etc.



```
"""
Author: Matthieu DE MARI

Description: computes and prints the determinant and the roots,
of a given quadratic equation ax^2 + bx + c = 0,
with strictly positive determinant.

Inputs: Requires the user to specify values of the parameters
a, b, and c. These should be numbers, ideally floats.

Outputs: this script calculates and prints the values of the
determinant and both roots.

Important note: if the determinant is not strictly positive,
then the roots will be incorrect.
"""


# ---------------------------------------
# 1. Define the coefficients (a,b,c) of
# the quadratic equation

a = 2
b = -2
c = -24

# ---------------------------------------
# 2. Compute the discriminant delta

delta = b**2 - 4*a*c

# ---------------------------------------
# 3.Print delta to check it is strictly positive

print(delta)

# ---------------------------------------
# 4. Compute the two roots (x1, x2)

x1 = (b + delta**0.5)/(2*a)
x2 = (b - delta**0.5)/(2*a)

# ---------------------------------------
# 5. Print the roots

print(x1)
print(x2)
```

Matt’s Great advice

Matt’s Great Advice: Comment your own code… Seriously… Do it.

It is always a good idea to **comment** your code, to let the reader know what your script does.

Either use **#** for **single line comments** or triple quotations (“””) for **block comments**.

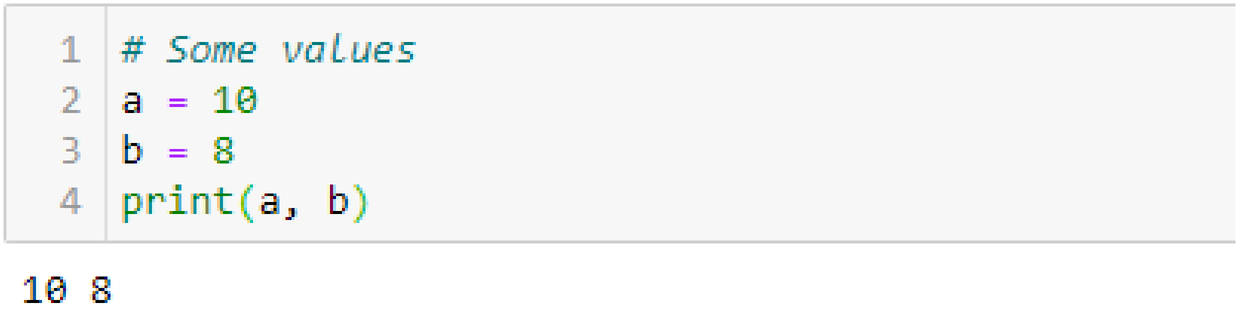
You can even use **separators (# ------)**, for readability, if needed.

Again, your future self and colleagues will be grateful, when they read code you have written.

About the **print()** function

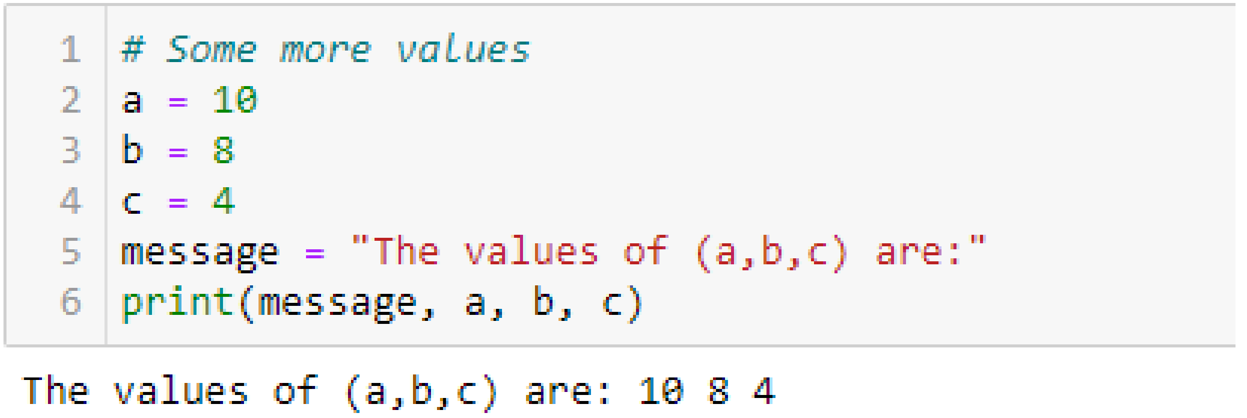
You can print multiple variables at once, using a single **print()**, by simply separating the different variables with commas.

- You can even mix several types of variables inside of a single print, using commas as a separator.
- The **print()** function always tries its best to show you the values of what you are requesting!



```python
# Some values
a = 10
b = 8
print(a, b)
```

```
10 8
```



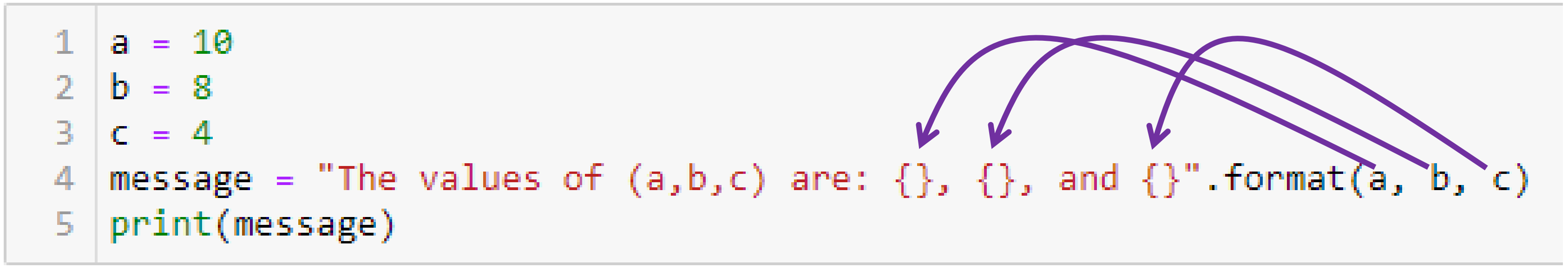
```python
# Some more values
a = 10
b = 8
c = 4
message = "The values of (a,b,c) are:"
print(message, a, b, c)
```

```
The values of (a,b,c) are: 10 8 4
```

About the **print()**, and the **format()** function

Definition (the format() method): The **format() method** can be used to **insert** the value of a variable in another string variable.

- It requires **some placeholders {}** in the string, and **some variables** passed to the **format() method**. Placeholders are simply replaced with the values of the variables passed to the **format() method**.
- **Note: format()** is a **special kind of function** called a **method**. Notice how it is applied to a string variable using the **dot operator**.



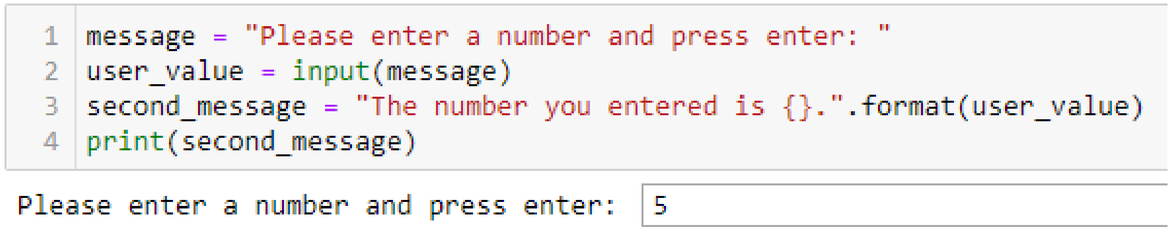
```python
a = 10
b = 8
c = 4
message = "The values of (a,b,c) are: {}, {}, and {}".format(a, b, c)
print(message)
```

```
The values of (a,b,c) are: 10, 8, and 4
```

Getting values from a user with **input()**

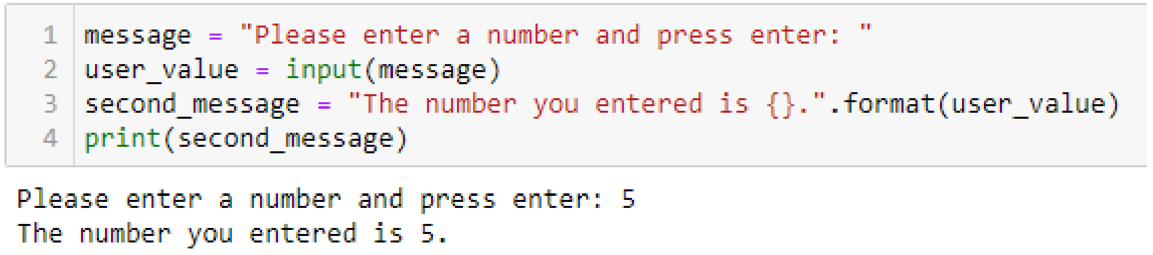
Definition (the input() function): You can also have Python ask the user for values explicitly. This is done with the **input()** function.

- Notice how it first **displays** the string variable message and a submission box,
- And later **assigns** the value typed by the user in the variable on the left-hand side of the equal sign used for input().



```python
message = "Please enter a number and press enter: "
user_value = input(message)
second_message = "The number you entered is {}.".format(user_value)
print(second_message)
```

```
Please enter a number and press enter:  5
```



```python
message = "Please enter a number and press enter: "
user_value = input(message)
second_message = "The number you entered is {}.".format(user_value)
print(second_message)
```

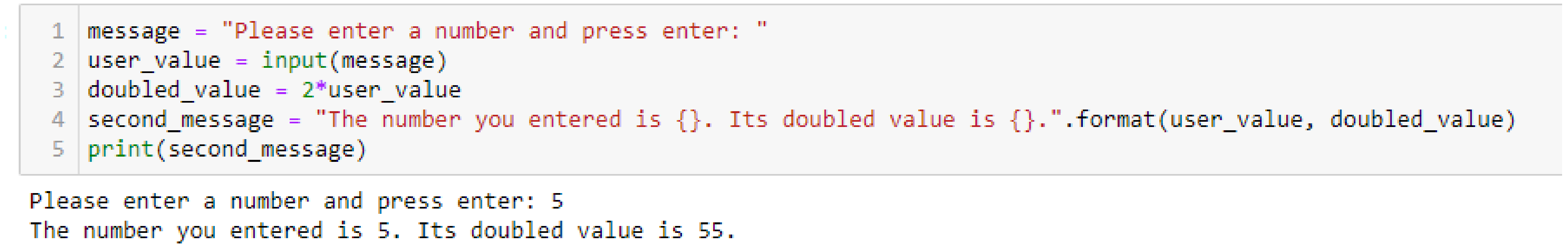
```
Please enter a number and press enter: 5
The number you entered is 5.
```

Getting values from a user with **input()**

Important: Be careful when getting values with **input()**!

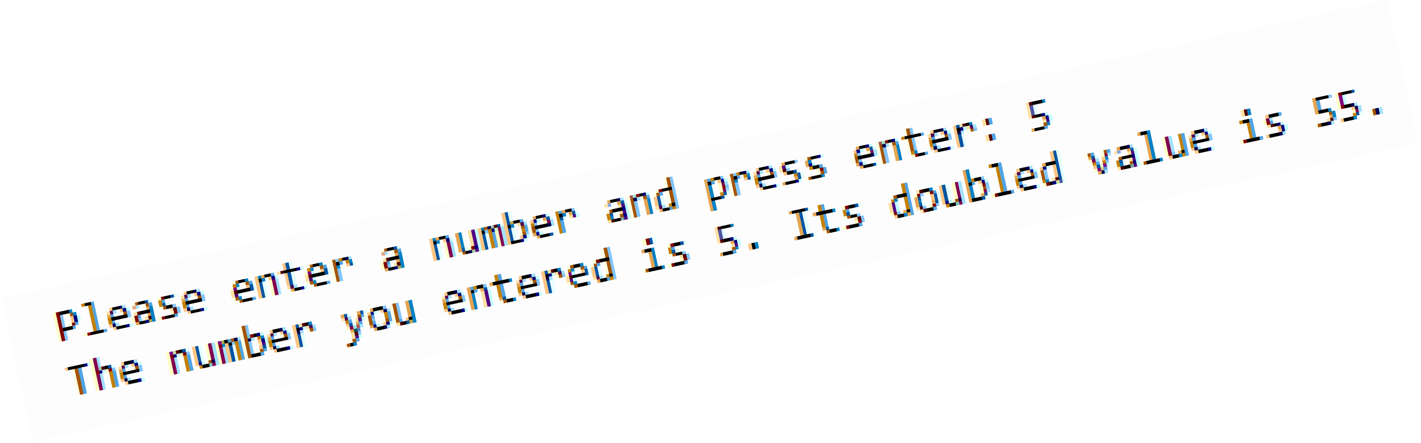
- In this program, I asked the user to enter a number,
- Doubled the value of said number,
- And later displayed both the original number and its doubled value.

Problem: Something strange occurs… But why?

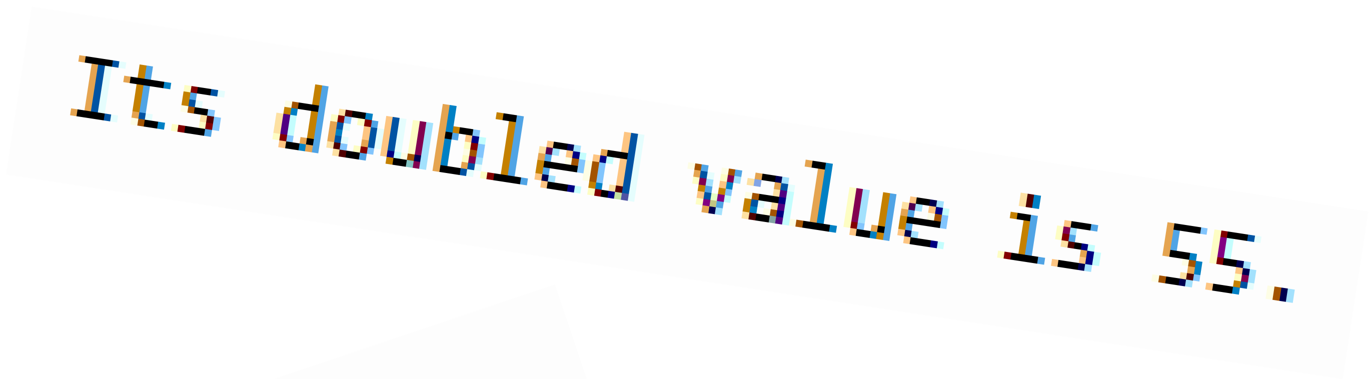


```python
message = "Please enter a number and press enter: "
user_value = input(message)
doubled_value = 2*user_value
second_message = "The number you entered is {}. Its doubled value is {}.".format(user_value, doubled_value)
print(second_message)
```

```
Please enter a number and press enter: 5
The number you entered is 5. Its doubled value is 55.
```



```
Please enter a number and press enter: 5
The number you entered is 5. Its doubled value is 55.
```



```
Its doubled value is 55.
```

```
55.
```

Has Python gone mad?

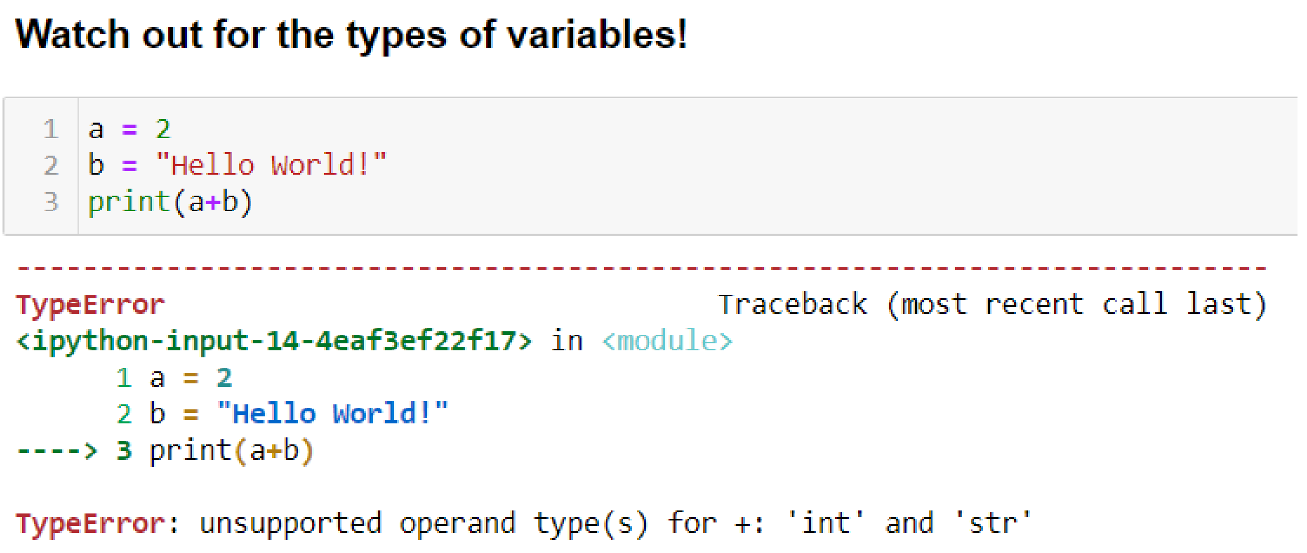
Getting values from a user with **input()**

Important note: The values retrieved from users with **input()** will always be typed as **string variables**!

Reminder: we do not do math with strings!

By default, Python comes with a few **basic math operators, for number (int/float) variables**.

- **+**: addition
- **-**: subtraction
- *****: multiplication
- **/**: division
- **//**: integer division
- **%**: remaining of the integer division (a.k.a. modulus)
- ******: exponentiation



Watch out for the types of variables!

```
a = 2
b = "Hello World!"
print(a+b)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-4eaf3ef22f17> in <module>
      1 a = 2
      2 b = "Hello World!"
----> 3 print(a+b)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Getting values from a user with **input()**

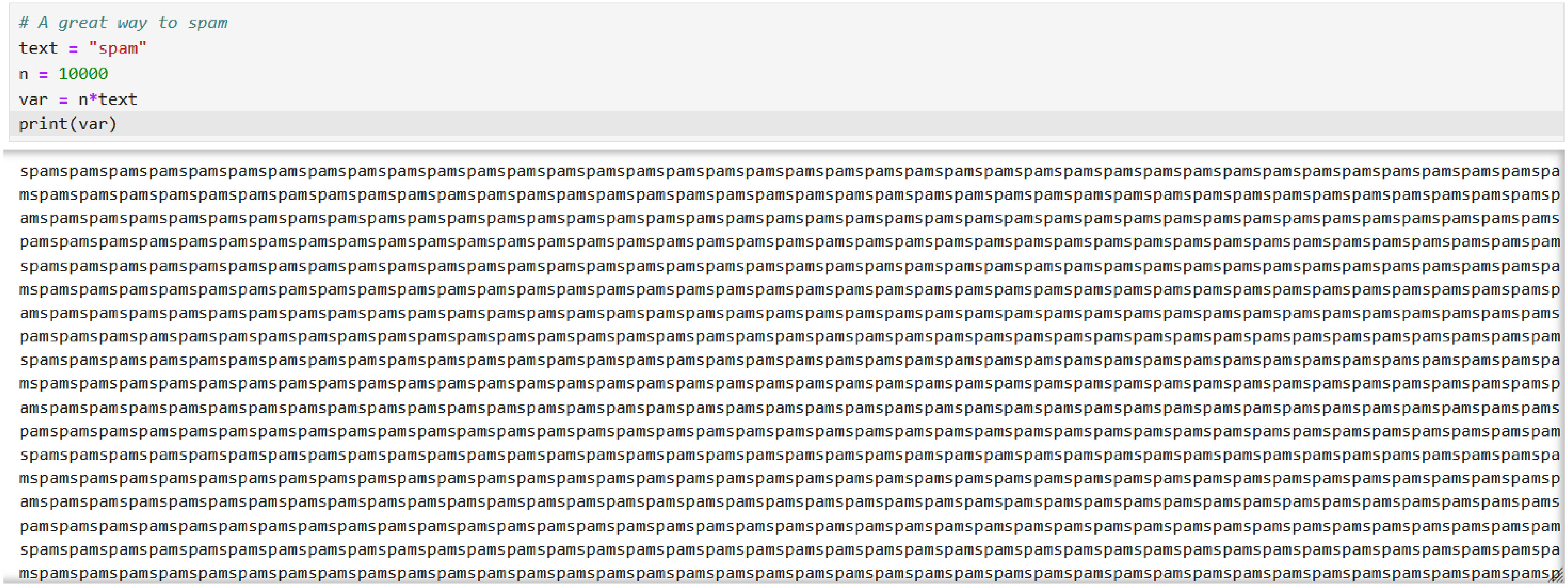
Important note: The values retrieved from users with **input()** will always be typed as **string variables**!

Fun fact: It turns out that multiplying a string “5” with an integer number n simply **repeats** the string n times!

- Hence, **“5”*2** is indeed **“55”.**

→ Solution: We need to convert our string variable to a float number variable <u>before</u> doing any math operations!

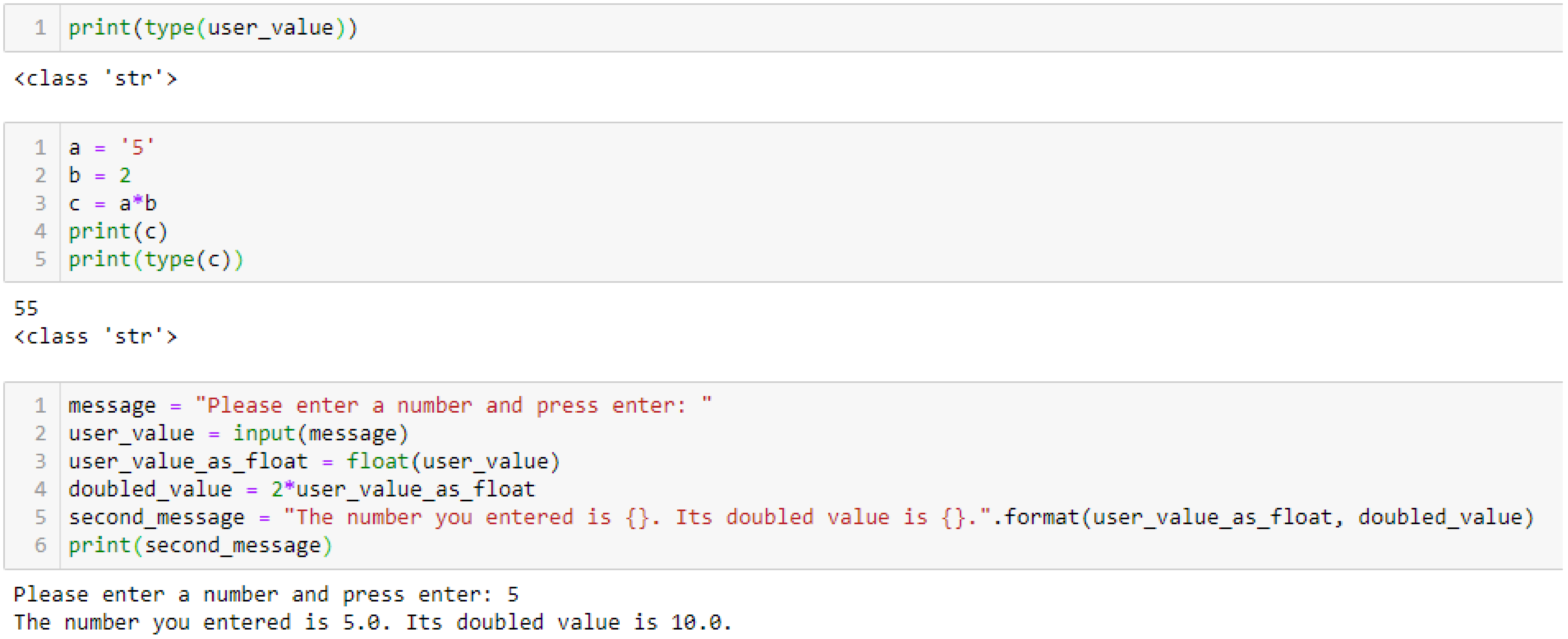
How to spam your enemies



```python
# A great way to spam
text = "spam"
n = 10000
var = n*text
print(var)
```

```
spamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspa
mspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamsp
amspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspams
pamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspam
spamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspa
mspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamsp
amspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspams
pamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspam
spamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspa
mspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamsp
amspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspams
pamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspam
spamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspa
mspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamsp
amspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspams
pamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspam
spamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspa
mspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamsp
```

Fixing our faulty code



```python
print(type(user_value))
```

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

```python
a = '5'
b = 2
c = a*b
print(c)
print(type(c))
```

```
55
<class 'str'>
```

```python
message = "Please enter a number and press enter: "
user_value = input(message)
user_value_as_float = float(user_value)
doubled_value = 2*user_value_as_float
second_message = "The number you entered is {}. Its doubled value is {}.".format(user_value_as_float, doubled_value)
print(second_message)
```

```
Please enter a number and press enter: 5
The number you entered is 5.0. Its doubled value is 10.0.
```

Matt’s Great advice

Matt’s Great Advice: Watch out for variables types with basic math operators!

The behavior of a basic math **operator** (+, -, *, etc.) might **change**, depending on the **types of the variables** involved in the calculation.

This is called **<u>operator overloading</u>** and is a perfectly normal feature of Python programming.

For instance, **multiplying a string and a number variables is not the same as multiplying two number variables!**

Activity 2: Ask users about personal details

Task: Write a Python script that explicitly asks the user for its name and its birth year using text prompts. Later on, the scripts should display a message, reading

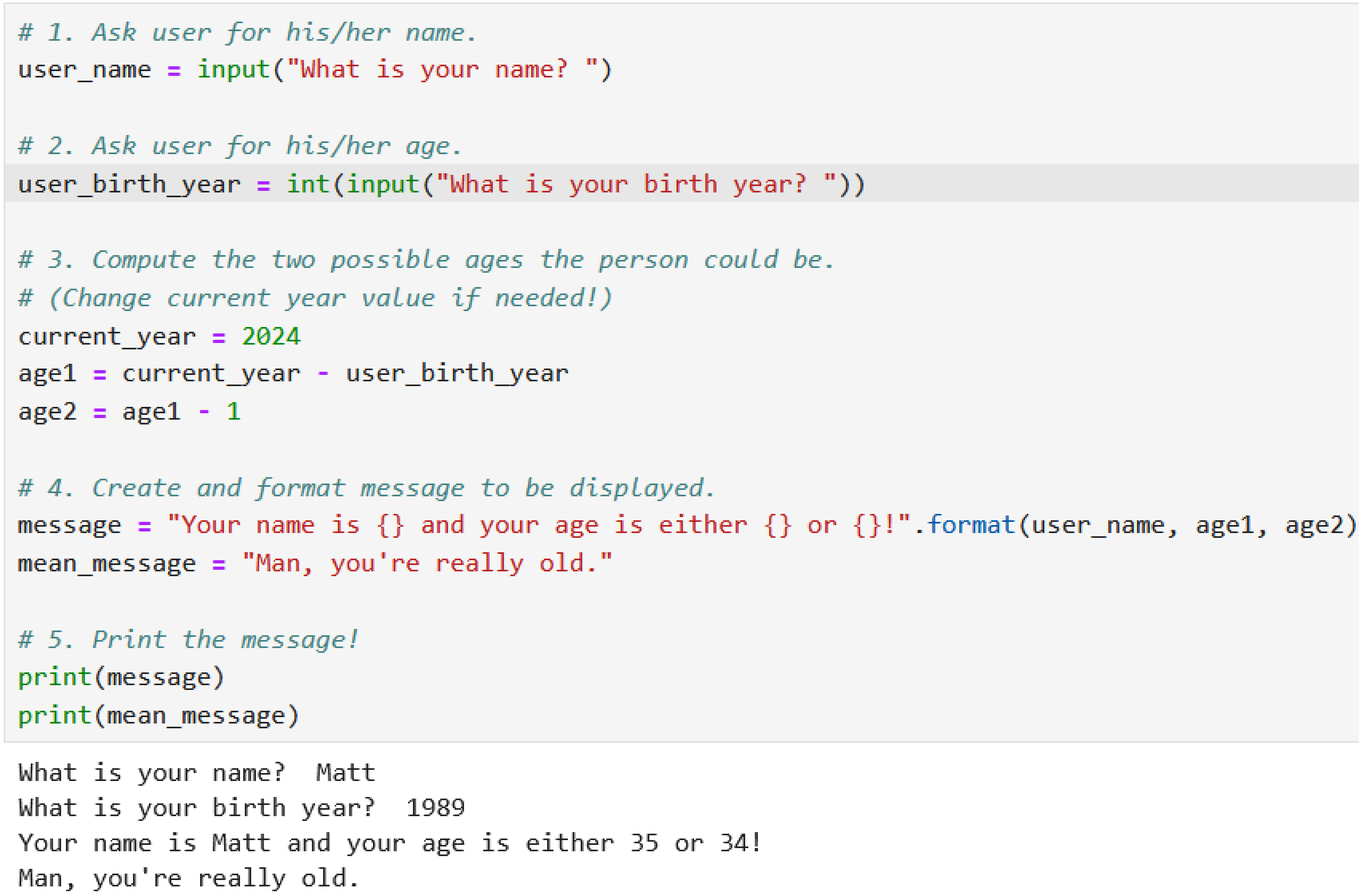
Your name is __________, and your age is either ____ or ____.

- You can use the second notebook

Activity 2 - Asking users for personal details.ipynb

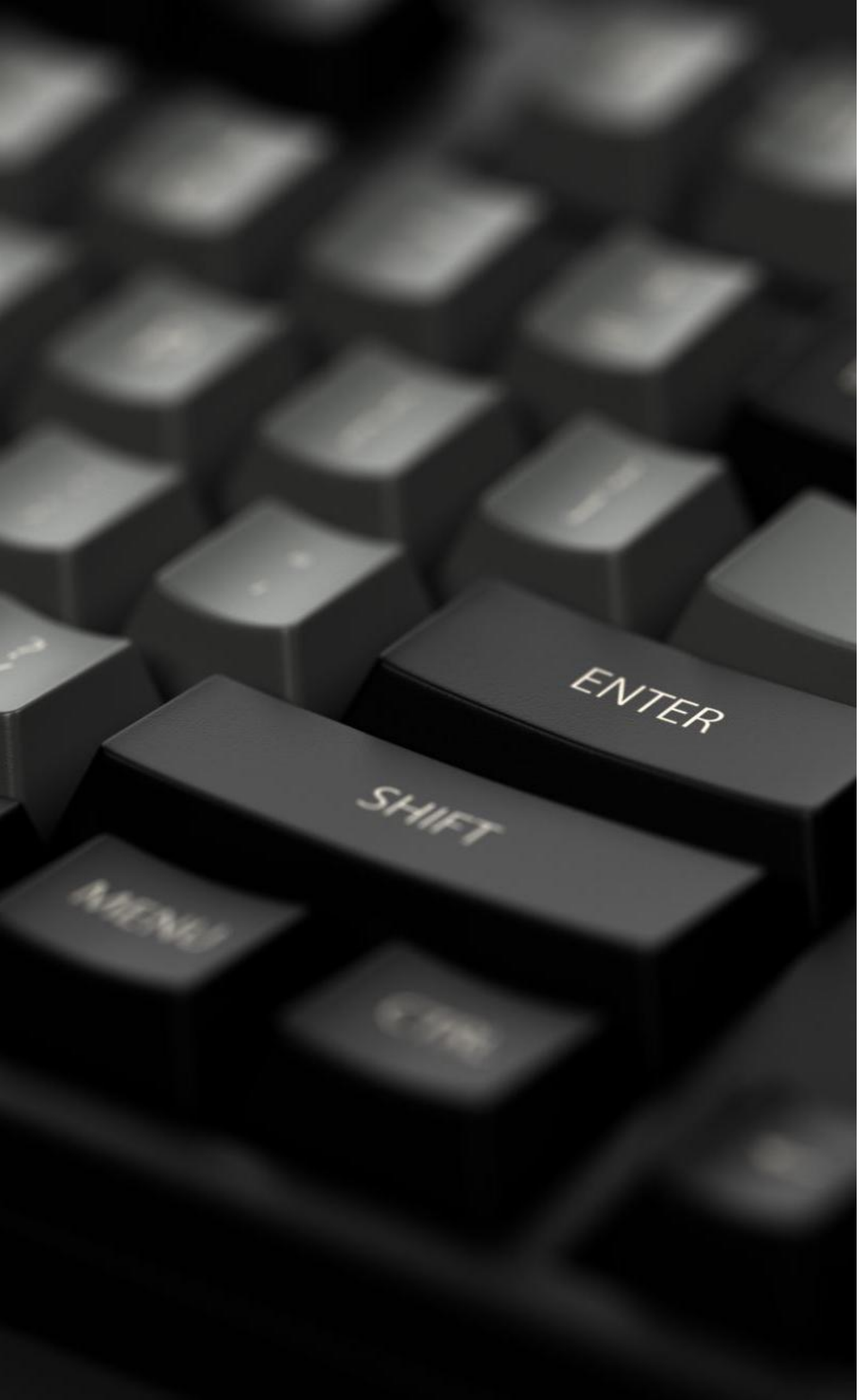
(If you finish early, have a look at the extra practice/challenges folder!)

Activity 2: Ask users about personal details



```python
# 1. Ask user for his/her name.
user_name = input("What is your name? ")

# 2. Ask user for his/her age.
user_birth_year = int(input("What is your birth year? "))

# 3. Compute the two possible ages the person could be.
# (Change current year value if needed!)
current_year = 2024
age1 = current_year - user_birth_year
age2 = age1 - 1

# 4. Create and format message to be displayed.
message = "Your name is {} and your age is either {} or {}!".format(user_name, age1, age2)
mean_message = "Man, you're really old."

# 5. Print the message!
print(message)
print(mean_message)
```

```
What is your name?  Matt
What is your birth year?  1989
Your name is Matt and your age is either 35 or 34!
Man, you're really old.
```



Conclusion (Chapter 2)

- **Everything you need to know about variables:** Naming conventions, types, IDs, assignments, types conversions.
- **Using basic math operators in Python on mathematical variables:** +, -, *, /, //, %, **.
- **Combining operators and assignments.**
- **Commenting:** in-line, block, docstring and header comments.
- **Printing and getting:** displaying with print(), getting with input().
- And more things!