Introduction to Python

Week 2 - IDEs, dict, comprehension and functions

Anaconda and Visual Studio Code

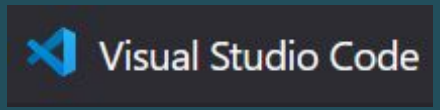
Turing Students ROTTERDAM

Python code needs to be interpreted before it is executed on the machine. Therefore a Python implementation is needed to run Python code on your computer.

Anaconda bundles Python with common packages for scientific computing, for example Jupyter Notebook, Numpy, Pandas, etc. It is recommended compared with just Python since it usually works out of the box without needing to change system settings.

Visual Studio Code is a Integrated Development Environment (IDE). It has many tools integrated in one interface to help with programming, for example a text editor with syntax highlighting, error checking and code formatting, and an integrated terminal.

We will be using this editor to write Python programs on the computer. After the installation, please install the Python plugin for the editor.



Dictionaries

Turing Students ROTTERDAM

Dictionaries are a collection of keys and values, and each key-value pair is an item. In python, dictionaries are written inside curly brackets {}.

```
population = {"Amsterdam":821752, "London":8136000, "Madrid":3174000}
```

You can access a value of a dictionary by its key:

```
print(population["Amsterdam"])
```

This line prints “821752”

Dictionaries

By default, iterating over a dict will give you the keys of that dict. But there are different ways of accessing the values and items of a dictionary:

- 1st for loop prints every key in population
- 2nd loop does the same in a clearer way
- 3rd loop prints the values
- 4th loop prints both keys and values

Turing Students ROTTERDAM

```
for i in population:
  print(i)

for k in population.keys():
  print(k)

for v in population.values():
  print(v)

for k, v in population.items():
  print(f"{k} has {v} people")
```

List comprehensions

Turing Students ROTTERDAM

List comprehensions provide a concise way of creating lists
For example, to create a list of the squared values from 0 to 9, one can do the following:

```
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
list_squared = []
for i in range(10):
    squared_value = i**2
    list_squared.append(squared_value)
```

With list comprehensions:

```
list_squared = [i**2 for i in range(10)]
```

List comprehensions

Turing Students ROTTERDAM

Looks familiar?

```
list_squared = [i**2 for i in range(10)]
```

$$\text{list squared} = \{x^2 \mid x \in \mathbb{N}, x < 10\}$$

Conditions can also be added:

```
# [0, 1, 4, 9, 16, 25]
list_squared_small = [i**2 for i in range(10) if i**2 < 30]
```

$$\text{list squared small} = \{x^2 \mid x \in \mathbb{N}, x < 10, x^2 < 30\}$$

List comprehensions

Turing Students ROTTERDAM

A string example:

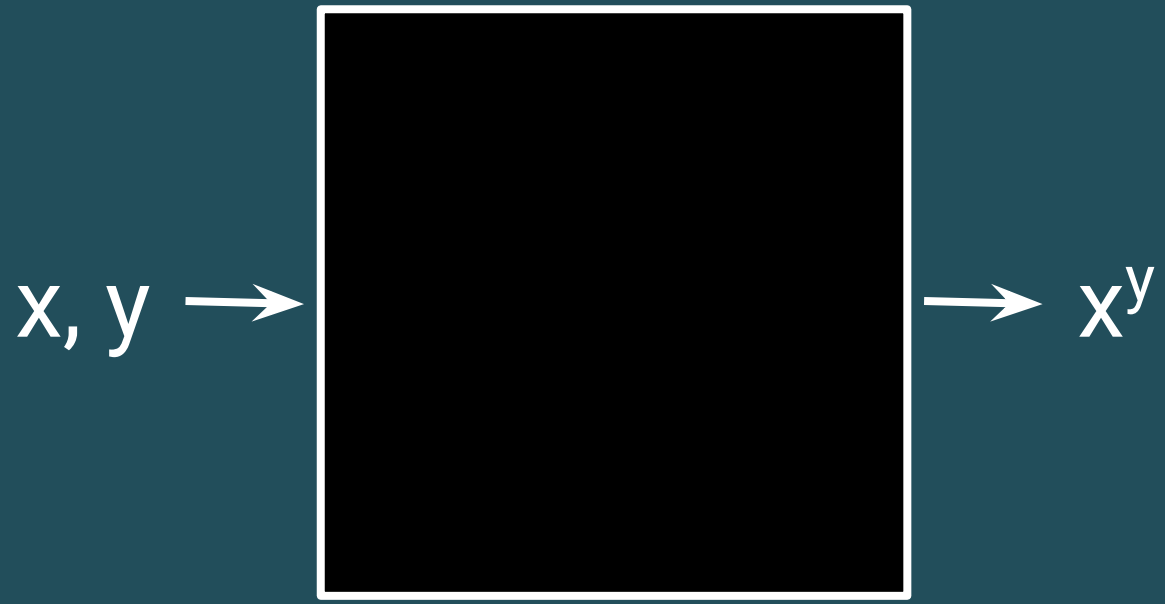
Suppose you want to make a list of the characters which are equal to a digit:

```
str_1 = "Th3 sum 0f th3 d1g1ts 3quals t0 th3 passw0rd"
list_string_code = [i for i in str_1 if i.isdigit()]
print("This is the password: ")
print(list_string_code)
```

This will print:
This is the password:
['3', '0', '3', '1', '1', '3', '0', '3', '0']

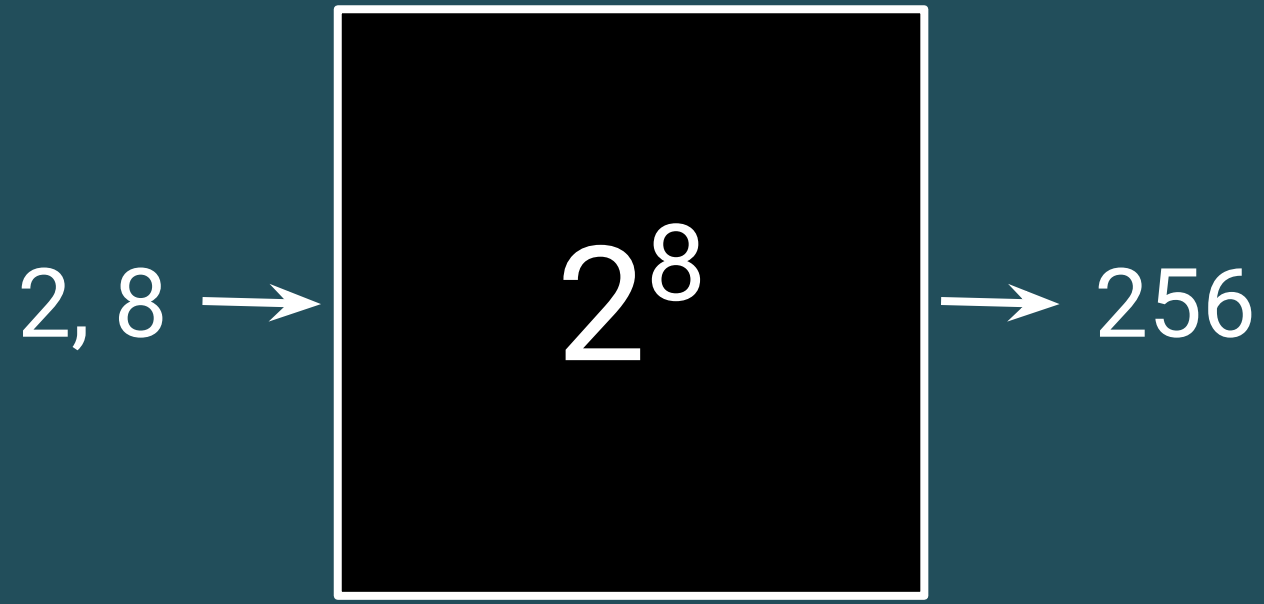
Functions

Turing Students ROTTERDAM



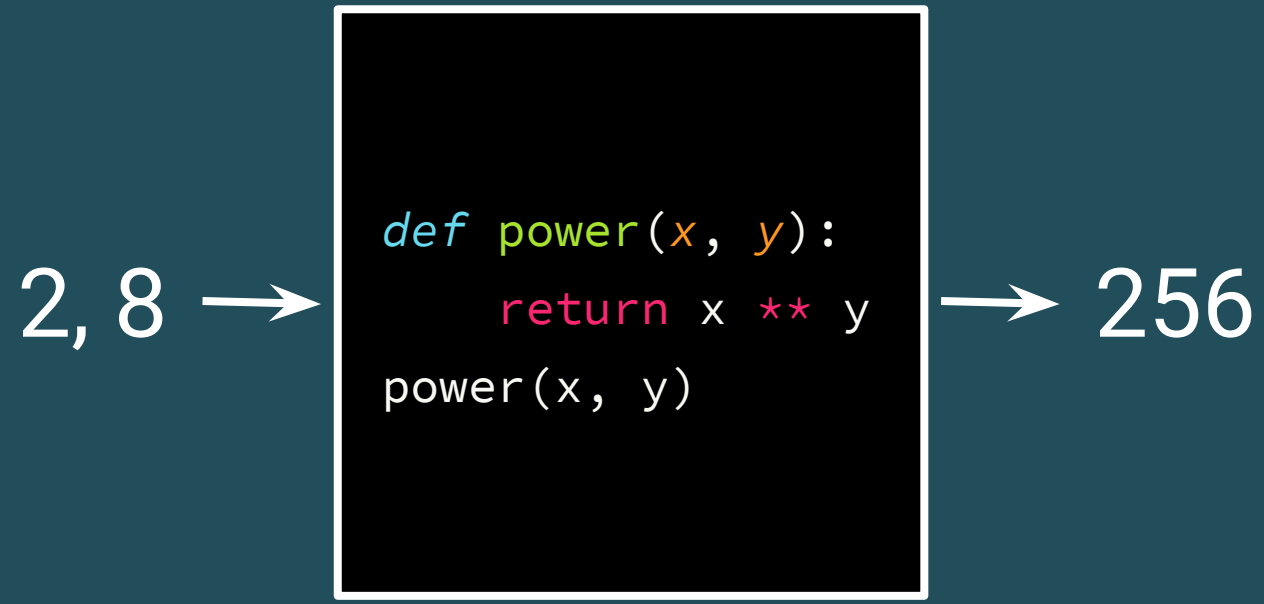
Functions

Turing Students ROTTERDAM



Functions

Turing Students ROTTERDAM



Functions

Turing Students ROTTERDAM

The main goals of a function are:

- Code reuse
  - Reduce the amount of repetitive code
  - Easier to change things in one place than everywhere in the file
- Separate a big program into smaller parts
  - Smaller blocks of code are usually easier to read
- Write cleaner code
  - Clearer input/output relationship makes code easier to reason about

Functions

Suppose you want to find the largest and smallest number of a list and then find the average of the two values.

You can do it like this, and repeat for every time you need to calculate the value:

Turing Students ROTTERDAM

```
list1 = [2,3,4,5,6]

min_value = min(list1)
max_value = max(list1)
average_min_max = (min_value + max_value)/2
print(f"The average of min and max is:
{average_min_max}")

list2 = [19,22,23,30,15]

min_value = max(list2)
max_value = min(list2)
average_min_max = (min_value + max_value)/2
print(f"The average of min and max is:
{average_min_max}")

...
```

Functions

```
def average_min_max(input_list):
    min_value = min(input_list)
    max_value = max(input_list)
    return (min_value + max_value)/2

list1 = [2,3,4,5,6]
list2 = [19,22,23,30,15]

print(f"The average of min and max is:
{average_min_max(list1)}")
print(f"The average of min and max is:
{average_min_max(list2)}")
```

Or this

```
list1 = [2,3,4,5,6]

min_value = min(list1)
max_value = max(list1)
average_min_max = (min_value + max_value)/2
print(f"The average of min and max is:
{average_min_max}")

list2 = [19,22,23,30,15]

min_value = min(list2)
max_value = max(list2)
average_min_max = (min_value + max_value)/2
print(f"The average of min and max is:
{average_min_max}")

...
```

TS Turing Students ROTTERDAM

Functions

Turing Students ROTTERDAM

How to create a function?

1. A name, just like a variable
2. Arguments
3. Code to be executed
4. (Optional) An output to return to the code calling the function

Turing Students ROTTERDAM

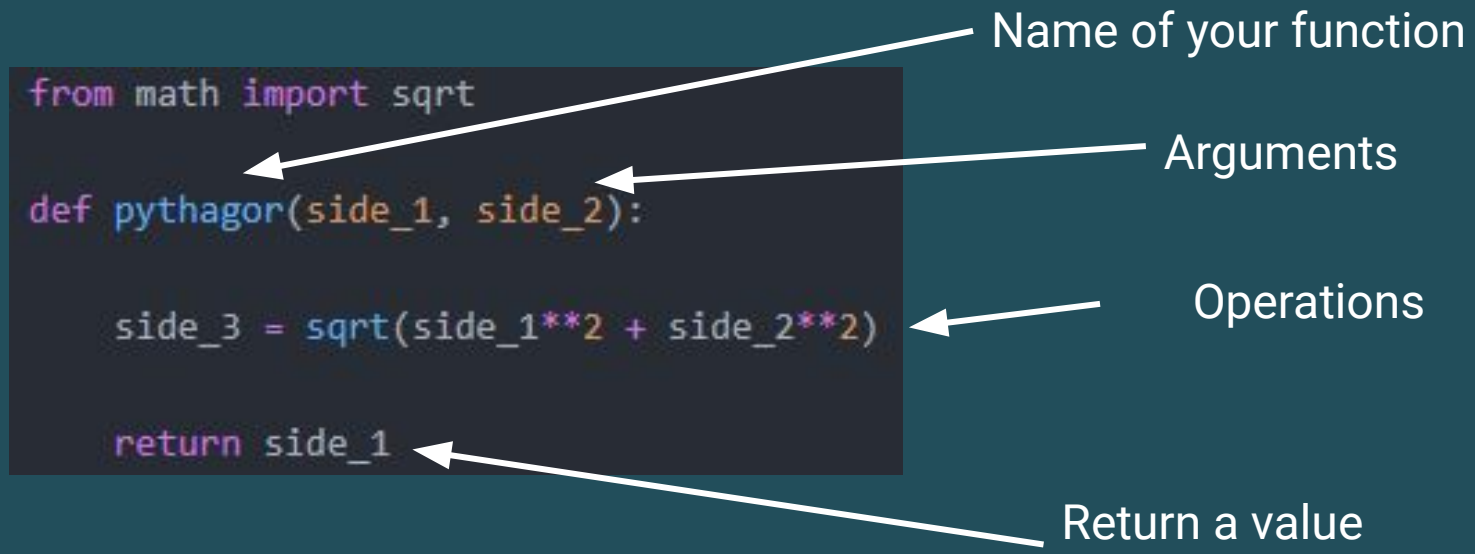
```python
from math import sqrt

def pythagoras(side1, side2):
    side3 = math.sqrt(side1**2 + side2**2)
    return side3

def pythagoras(side1, side2):
    return math.sqrt(side1**2 + side2**2)
```

Functions

Turing Students ROTTERDAM



Functions

Turing Students ROTTERDAM

Important remark!

Functions have their own variable scope. What this means is that the variables INSIDE a function will remain inside and can’t be used outside the function's body. For example:

Functions

```
def math_operation(numb_1, numb_2):

    x = numb_1/numb_2

    numb_3 = numb_1 + numb_2

    return numb_3

n = math_operation(numb_1 = 4, numb_2 = 2)

print(n)
print(x)
```

This will print:
6

and then it will give an error since x is not returned from the function.

Turing Students ROTTERDAM

Functions

What do you think will happen now?

Turing Students

```
x = 5

def math_operation(numb_1, numb_2):

    x = numb_1/numb_2

    numb_3 = numb_1 + numb_2

    return numb_3

n = math_operation(numb_1 = 4, numb_2 = 2)

print(n)
print(x)
```

Functions

Turing Students ROTTERDAM

You can return multiple variables from a function:

```
def my_func(a, b, c):

    var1 = a
    var2 = a + b
    var3 = a + b + c

    return var1, var2, var3

x, y, z = my_func(1,2,3)
print(x)
print(y)
print(z)
```

This will print:
1
3
6

Functions

```
def square_sum(a , b):

    c = (a + b)**2

print(c)
```

This will give an error since the variable "c" inside the function is not returned

Turing Students ROTTERDAM

Functions

```
def square_sum(a , b):

    c = (a + b)**2

    return c

var1 = square_sum(a = 1, b = 2)
print(var1)
```

Now since the variable "c" inside the function is returned and assigned to var1, one can make use of it.

This will print:
9

Turing Students ROTTERDAM