

# Introduction to Python Day 2

Verjina Metodieva and Daniel Parthier

2025-02-04

## Jupyter Notebook

### Recap homework

Let's take a look at the homework

### Functions part 2

### Goal of today

```
import numpy as np
import os

def AP_check(folder):
    AP_sweep_count = 0
    for filename in os.listdir(folder):
        if filename.endswith('.csv'):
            with open(os.path.join(folder, filename), 'r') as file:
                data = np.loadtxt(file, skiprows=1)
                if any(data > 20):
                    AP_sweep_count += 1
                    print("AP found in " + filename)
                else:
                    print('No AP in ' + filename)
    return AP_sweep_count

file_path = 'data/sweeps_csv/'
```

```
sweep_count = AP_check(file_path)
print(sweep_count)
```

---

This is an example to showcase what we will achieve today.

## Global vs. Local

- Scopes
  - Local variables live and die inside a function
  - Global variables
    - declared outside of functions
    - lost when programme closed
- 
- global scope - of the whole program
  - local scope of a separate function
  - Local variables
    - Variables that live only inside of functions, minutes in our example
    - They no longer exist once the function is done executing show it by running minutes
    - If we try to access their values outside the function, we will encounter an error
  - Global variables
    - variables defined outside any function

## Short interlude

- Whole numbers: Integers `int`

```
type(1)
```

```
int
```

```
number_string = "1"
number = int(number_string)
print(number)
type(number)
```

1

int

- Real numbers: Floats float

```
type(1.0)
```

float

```
fake_integer = float(1)
type(fake_integer)
```

float

- Most of the time it might not matter<sup>1</sup>

```
1 == 1.0
```

True

- Sometimes there is a difference, and we will see later why

---

Most of the time python handles the integer vs. float automatically. You will not have to worry about assigning.

### But sometimes it does

- Sometimes you will see `is` instead of `==`.

```
1 == 1.0
```

```
1 is 1.0
```

```
<>:2: SyntaxWarning: "is" with 'int' literal. Did you mean "=="?
```

```
<>:2: SyntaxWarning: "is" with 'int' literal. Did you mean "=="?
```

True

---

<sup>1</sup>In python

```
/tmp/ipykernel_2228/3921864666.py:2: SyntaxWarning: "is" with 'int' literal. Did you mean "="?
1 is 1.0
```

False

- They often do the same thing but work differently
    - Careful when using it!
    - `is` and `is not` is also checking the type!
- 

This can sometimes lead to an unexpected behaviour if you did not plan to use it in this way. Only use `is` and `is not` when you really want the identical object.

## Conditional statements

The important question of what to do “if” something happens.

- Programming languages are languages
- `if` something is `True`
  - you should do `something`
- `else`
  - do `something else`

```
if *statement*:
    print("the *statement* is true")
else:
    print("the *statement* is false")
```

---

This structure is the simplest of conditionals. The statement has to be `True` to enter the `if` part to execute. Should the statement be `False` it will skip and enter the `else` part which will then be executed.

## Multiple if-statements

```
value = 3
if value == 1:                                ①
    print("the value is 1")
elif value == 2:                              ②
    print("the value is 2")
elif value == 3:                              ③
    print("the value is 3")
else:                                         ④
    print("the value is something else")
```

- ① Check if value is 1
- ② Check if value is 2
- ③ Check if value is 3
- ④ Execute block

```
the value is 3
```

---

Statements will be checked sequentially. Should one statement be **True** the corresponding part of the **if/elif** block will be executed. All other blocks after that will be skipped. This means one **True** expression is enough.

## Short forms for conditionals

```
amplitude = 24
is_action_potential = "is AP" if amplitude > 0 else "no AP"
print(is_action_potential)
```

```
is AP
```

- You can write a lot on one line
  - Do if you have to but be careful

## How to check if everything is true?

- Validate all the statements in a list

```
everything_is_true = [True, True, True]
something_is_true = [True, False, False]
```

```
all(everything_is_true)
all(something_is_true)
```

True

False

- Sometimes only something has to be true

```
any(everything_is_true)
any(something_is_true)
```

True

True

## For loops

```
for *element* in *iterable*:  
    *body*
```

- iteration is the repetition of a process until a specific condition is met
- what's iterable?

```
# calculate a sum  
list_to_sum = [2,3,4,5,7]  
num_sum = 0  
for val in list_to_sum:  
    num_sum = num_sum + val
```

- 
- For loop = An iterating function used to execute statements repeatedly.
  - Iterate = In programming, iteration is the repetition of a code or a process until a specific condition is met.
  - Iterables = objects in Python that you can iterate over, e.g. container types (list, numpy arrays, tuples, sets), dictionary.keys(), strings

## TO DO

Given: A = [3, 4, 5, 9, 12, 87, -65, 300, 450, -32]

Use *for loops* to: 1. Add 3 to each element of the list 2. Calculate the average of the list, but negative values are calculated as 0s 3. Find the maximum value 4. Find the index (position) of the max value

### Index based *for loops* - range()

- generates integer sequences
- range(n) generates the series of n values from 0 to n - 1

```
for i in range(5):  
    print(i)
```

```
0  
1  
2  
3  
4
```

```
# looping through data indices. find the max  
B = [1, 4, 6, 7, 89, 54]  
big_indx = 0  
for i in range(len(B)):  
    if B[i] > B[big_indx]:  
        big_indx = i  
print('The max value in B is', B[big_indx], 'found on position', big_indx)
```

The max value in B is 89 found on position 4

### Index based *for loops* - enumerate()

- assigns a count to each item within an iterable and returns it as an enumerate object

```
import numpy as np  
  
array_a = np.arange(20, 25)  
for indx, val in enumerate(array_a):  
    print('the index is', indx)  
    print('the value is', val)
```

```
the index is 0
the value is 20
the index is 1
the value is 21
the index is 2
the value is 22
the index is 3
the value is 23
the index is 4
the value is 24
```

! range() and enumerate() - none of the two returns a list of objects!

- 
- motivation: limitation in 'simple' for loops - we don't know the position of an element within a sequence, as we experienced in the last example
  - range(n) generates the series of n values from 0 to n - 1
  - precisely the series of valid indices into a sequence of length n
  - range() - returns a range object that is iterable
  - enumerate() - returns an enumerate object that is also iterable
  - they are mainly used in loops

## Index based *for loops* [1]

### enumerate()

- one way to avoid nested loops

## Break and continue statements

- break - immediately terminates the loop

- 
- a break statement that immediately terminates a while or for loop when executed within its body
  -



## List comprehension

## Compare different functions

## While loops

- Perform a task `while` something is `True`
- Be careful:
  - Some loops never finish (get stuck)
  - Make sure that condition for ending the loop can be fulfilled

```
while check_condition:  
    perform_task()
```

---

If your python terminal gets stuck at one point you can try a `KeyboardInterrupt` using `Ctrl+C`, which will kill the running script.

## Let's wait while we wait

- Start a little counter

```
import time  
counter = 0  
while counter < 10:  
    time.sleep(1)  
    counter += 1  
    print("You waited for " + str(counter) + " seconds...")
```

- Good for keeping processes running

Try to avoid `while` loops as much as possible. They can be useful if you do not have information how long it should run, but know it will at one point finish.

## Errors and how to read them

There are useful resources regarding errors

- Simply googling works surprisingly well
- You will often end up on [stackoverflow](#)
  - There is no question which was not already asked<sup>2</sup>

The screenshot shows the Stack Overflow search results page for the query "if clause python". The page has a sidebar on the left with navigation links like Home, Questions, Tags, Users, Companies, LABS, Jobs, Discussions, COLLECTIVES, and TEAMS. The main content area displays search results with 500 results found. The top results include:

- Manually raising (throwing) an exception in Python** (4281 votes, Accepted). Problem 1: Hiding bugs raise Exception("I know Python!") # Don't! If you catch, likely to hide bugs. ... Best Practices: except clause When inside an except clause, you might want to, for example, log th...
- Why does python use 'else' after for and while loops?** (778 votes, 25 answers). I'm wondering how Python coders read this construct in their head (or aloud, if you like). Perhaps I'm missing something that would make such code blocks more easily decipherable? ... See also Else...
- What does if \_\_name\_\_ == "\_\_main\_\_": do?** (2200 votes). When your script is run by passing it as a command to the Python interpreter, python myscript.py all of the code that is at indentation level 0 gets executed. ... If your script is being imported into another...
- Python Dictionary if clause [duplicate]** (-1 votes, 2 answers). as you can tell, I just recently started to learn python. While practising, I stumbled upon an issue, for which I don't have an answer to why this is the case. ... Saigon are not in the dictionary Python no...
- How to execute multi-line statements within Python's own debugger (PDB)** (248 votes, 6 answers). Now, when I get to the debugger I want to execute a multi-line statement such as an if clause or a for loop but as soon as I type if condition: and hit the return key, I get the error message \*\*\* SyntaxError ...
- executing wrong if-clause in python [duplicate]** (-1 votes).

On the right side of the page, there are two job advertisements from Indeed and a section titled "Hot Network Questions" with various programming-related queries.

## Types of errors

1. SyntaxErrors
2. IndentationError
3. NameError
4. TypeError
5. IndexError
6. AttributeError
7. etc.

---

<sup>2</sup>if that is not true open up a question

---

### **Fix errors**

- Breathe
- Don't panic
  - Identify the error by checking the terminal output
  - Look at the line provided
  - Go backwards if error is nested