University of Zurich[UZH]

Department of Molecular Life Sciences

Introduction to Python

BIO334

Maria Heimlicher & Matteo Peluso & Tülay Karakulak

Christian von Mering group

maria.heimlicher@mls.uzh.ch

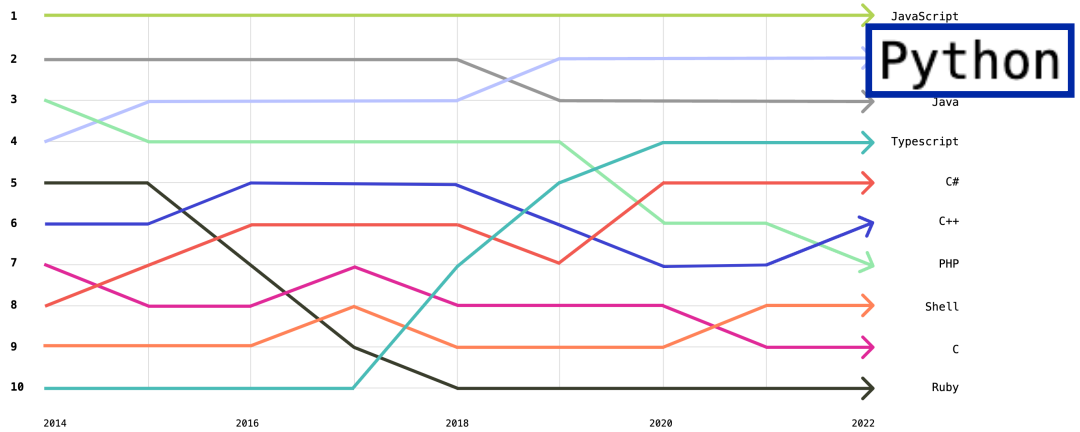
courtesy of Danaila, Sumner, Lyon, Tackmann, Dmitrijeva & Gable

https://github.com/meringlab/Bio334.git

What is Python?

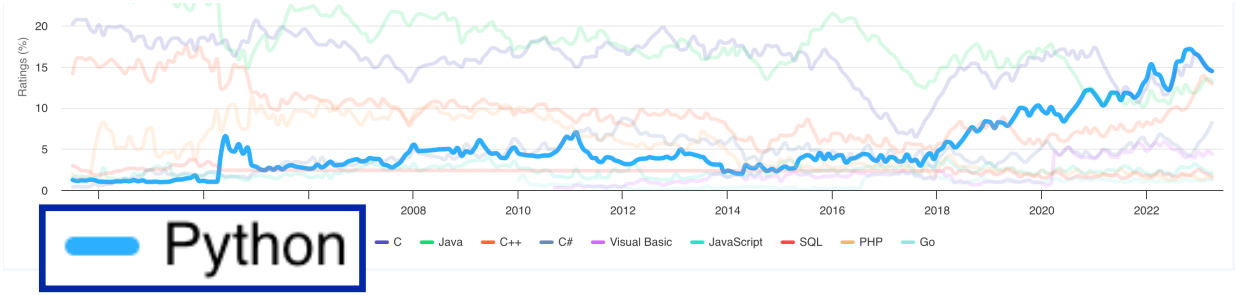
- Dynamic, interpreted, open source **programming language**
- **Simple syntax** with fast learning curve

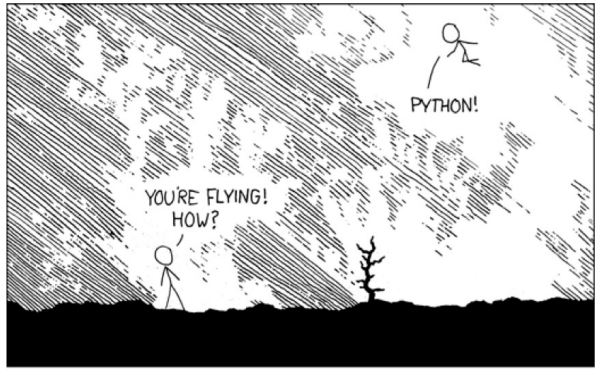
Octoverse by Github: Top languages used in 2022



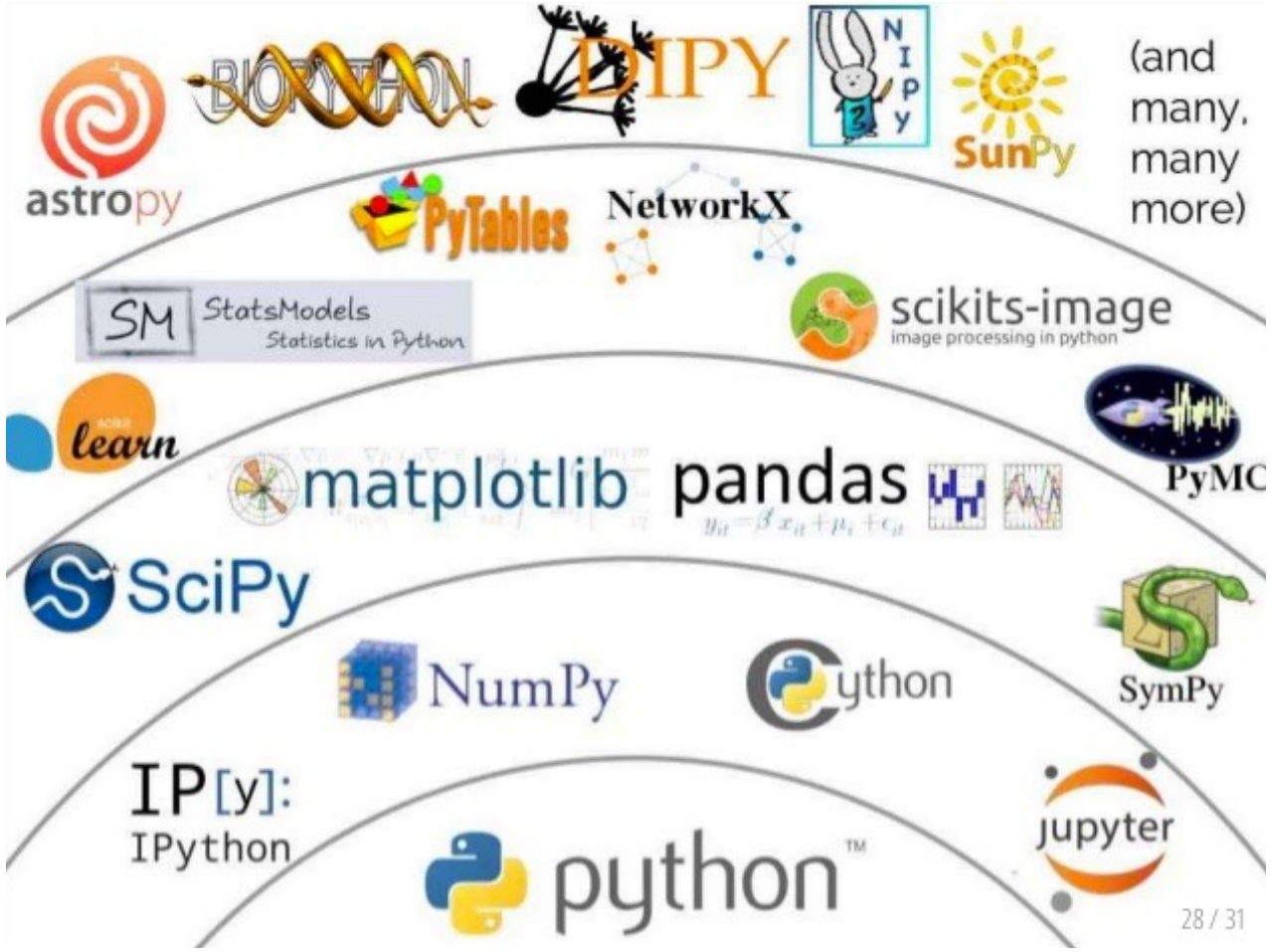
Stack Overflow 2022: most popular languages

TIOBE Programming Community Index





Scientific Python stack: more than core Python



Today's program

1. Introduction to programming:
   refresh the basic concepts with small hands-on sessions in JupyterLab
2. Break
3. Writing Python code to solve Exercises 1 and 2 (and optional Exercises 3 and 4)
4. Optional: go through solutions of exercise 1 and 2 at **4pm**

Part 1:
Programming basics

Variables

Store a piece of data and give it a **specific name** with the "**=**" (equal) operator

```
a = 4
pi = 3.14159

my_string = "hello"
# single and double ticks are equal
# (just stick to one) 'hello' or "hello"

my_protein_sequence  = 'MRHIAHTQRCLSRLTSLVALLLIVLP'
```

Basic **rules** for variable names:

- NO spaces
- don't start with numbers
- use descriptive names

Operators

Arithmetic operations

- addition + , subtraction -
- division /, multiplication *
- exponent **

```
>>> 2 + 3
5
>>> 23 - 3
20
>>> 22.0 / 12
1.8333333333333333
>>> (1 + 2) * 3
```

Boolean operations

- return True or False
- comparison operators: == , > , <
- logical operators: and, or, not

```
>>> 1 < 2
True
>>> 23 == 45
False
>>> 2<3 and 4!=3
True
```

Get ready for live sessions in JupyterLab: Download course files

1. open a **Terminal window** (e.g. Spotlight search: terminal):
2. change to your personal course folder (or where you would like the files to go)

   ```
   > cd /Volumes/Kurs/UsersBio/mheiml  #replace with your shortname
   ```

3. clone the git repository by entering:

   ```
   > git clone https://github.com/meringlab/Bio334.git
   ```

- checkout our course files under *02_python_introduction*
- next up: installation and short demo on how to use JupyterLab

Get ready for live sessions in JupyterLab: (Install and) launch JupyterLab

open a **Terminal window** and enter the following commands

```
> cd /Volumes/Kurs/UsersBio/yourShortname
> . /opt/miniconda3/etc/profile.d/conda.sh #space after dot matters!
> conda create --prefix /Volumes/Kurs/UsersBio/yourShortname/py38_envs
  python=3.8 #copy whole line starting from conda
> conda activate /Volumes/Kurs/UsersBio/yourShortname/py38_envs
> conda install jupyter #press y and Enter when asked
> jupyter lab #to open jupyter lab
```

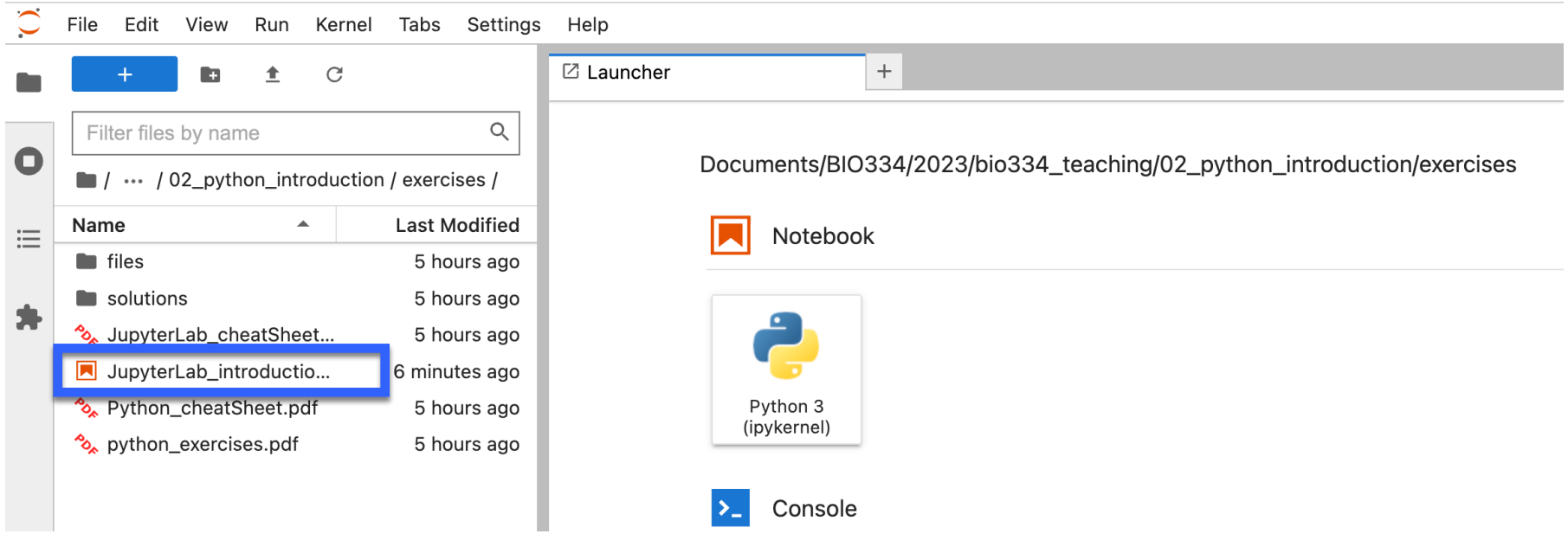
first time: run all the commands, blue and yellow

after this: run only the blue commands to open JupyterLab again

Get ready for live sessions in JupyterLab: start your first Jupyter Notebook

- in Jupyter: go to *Bio334/02_python_introduction/exercises*

- launch the Jupyter Notebook *JupyterLab_introduction.ipynb* for an interactive session



Session I: variables and basic operations

```
welcome_message = 'hello world!' # hit [shift + enter] after every line
welcome_message # more general alternative: print(welcome_message)

# Arithmetics: use python as your advanced calculator
a = 4
b = a + 3
(a + b) * 4
a / 8
a**2

# Let's try with strings
welcome_message + welcome_message

a = '4'
a + 3 # anything strange?

# whats the difference?
a + str(3)
int(a) + 3
```

Discussion session I: Arithmetics

[1]:
```
welcome_message = 'hello world!' # hit [shift + enter] after every line
```

[2]:
```
welcome_message
```

[2]:
```
'hello world!'
```

[3]:
```
# Arithmetics: use python as your advanced calculator
a = 4
```

[4]:
```
b = a + 3
```

[5]:
```
(a + b) * 4
```

[5]:
```
44
```

[6]:
```
a/8
```

[6]:
```
0.5
```

[7]:
```
a**2
```

[7]:
```
16
```

Discussion session I: Concatenation

[8]:
```
# Let's try with strings
```

[9]:
```
# use + to add two strings
welcome_message + welcome_message
```

[9]:
```
'hello world!hello world!'
```

[10]:
```
a = '4'
```

[11]:
```
# We need to have the same type to add elements
```

[12]:
```
a + 3 # anything strange?
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/3s/6lmkv5_10g1g4vx768t1yh_0g4tfkb/T/ipykernel_50845/173162209.py in <module>
----> 1 a + 3 # anything strange?

TypeError: can only concatenate str (not "int") to str
```

[ ]:
```
# whats the difference?
```

[13]:
```
a + str(3)
```

[13]:
```
'43'
```

[14]:
```
int(a) + 3
```

[14]:
```
7
```

List data structure

- like a shopping list we have an object that can store multiple objects at once.

```
my_list = ['butter','milk','oranges']
```

- it can hold different objects as well

```
my_list = ['butter',1,1.5,'milk']
```

- every element has an index, starting from 0 which you can access with the square brackets [ ]

```
>>> my_list[0]
'butter'
```

List data structure

- get subset of a list (“slicing”)

```
>>> my_list[0:2]
['butter','milk']
```

- assign new values to list elements

```
my_list[0] = 'bananas'
```

- powerful list operations
  - e.g. append(), len(), sort(), reverse(), insert()

Session II: Work with lists

```
# create your first list
# try accessing it with indices
my_list = [1,2,3,4,5]
my_list[1]
my_list[5]
my_list[0:3]
my_list[-1]

len(my_list)

# mixed lists:
# put different data types in your list

my_mixed_list = ['UZH','founded in',1934]
my_mixed_list[2] = 1834
del my_mixed_list[1]
my_mixed_list.append('A.D.')

my_mixed_list
```

```
# apply arithmetic operators on lists

[1,2,3] + [3,4,6] # list concatenation

['Hello'] * 4
```

Discussion session II: Accessing list elements

[22]:
```
# create your first list and try accessing it with indices
my_list = [1,2,3,4,5]
```

[18]:
```
my_list[1]
```

[18]:
```
2
```

[19]:
```
my_list[5]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/var/folders/3s/6lmkv5_10g1g4vx768t1yh_0g4tfkb/T/ipykernel_50845/303719451.py in <module>
----> 1 my_list[5]

IndexError: list index out of range
```

[42]:
```
my_list[0:3] # slice your list with index ranges
```

[42]:
```
[1, 2, 3]
```

[44]:
```
my_list[-1] # get the last element with the index -1
```

[44]:
```
5
```

[45]:
```
len(my_list) # check how many elements your list contains
```

[45]:
```
5
```

Discussion session II: List modifications and arithmetic operations on lists

[55]:
```
# mixed lists, put different variable types in your list
my_mixed_list = ['UZH','founded in',1934]
```

[56]:
```
my_mixed_list[2] = 1834
```

[57]:
```
del my_mixed_list[1] # delete an element at a specific index
```

[58]:
```
my_mixed_list.append('A.D.') # add a new item to the end of the list
```

[59]:
```
my_mixed_list
```

[59]:
```
['UZH', 1834, 'A.D.']
```

[60]:
```
# apply the arithmetic operators on lists
[1,2,3] + [3,4,6]
```

[60]:
```
[1, 2, 3, 3, 4, 6]
```

[61]:
```
['Hello'] * 4
```

[61]:
```
['Hello', 'Hello', 'Hello', 'Hello']
```

Strings & Lists

- Strings can also be considered lists of characters and can be accessed by index
- To convert them to an actual list (and make them mutable) use list()

```
>>> my_sequence = 'MRHIAHTQRCLSR'
>>> list(my_sequence)
['M', 'R', 'H', 'I', 'A', 'H', 'T', 'Q', 'R', 'C', 'L', 'S', 'R']
```

- While this opens many possibilities, check the cheat-sheet for convenient built-in string operations
  - e.g. split(), join(), replace()

Dictionaries

- Similar to lists, but elements are accessed through a user defined key

```
my_proteins_seqs = {}
my_proteins_seqs = dict() # same as above
my_proteins_seqs['DROME_HH_Q02936'] = 'MRHIAHTQRCLSRLTSLVA'
my_proteins_seqs['DROME_PATC_P18502'] = 'MDRDSLPRVPDTHGDVVD'
```

- retrieve their **value** by using the **key**

```
>>> my_proteins_seqs['DROME_HH_Q02936']
'MRHIAHTQRCLSRLTSLVA'
```

Session III: Strings and dictionaries

```
# Try accessing a string like a list

my_string = 'hello world!'

my_string[6]
my_string[-1] = '?'

my_list = list(my_string)
my_list[-1] = '?'
my_modified_string = ''.join(my_list)

my_modified_string
```

```
# create your first dictionary

dna_to_rna = {}
dna_to_rna['A'] = 'A'
dna_to_rna['T'] = 'U'
dna_to_rna['C'] = 'C'
dna_to_rna['G'] = 'G'
dna_to_rna

#or in one line
dna_to_rna =
{'A':'A','T':'U','C':'C','G':'G'}

#dictionary operations

dna_to_rna['U']
'U' in dna_to_rna
dna_to_rna.keys()
dna_to_rna + dna_to_rna
```

Discussion session III: String lists

[65]:
```
# Try accessing a string like a list
my_string = 'hello world!'
```

[72]:
```
my_string[6] # accessing a character by index
```

[72]:
```
'w'
```

[73]:
```
my_string[-1] = '?' # changes are not allowed - strings are immutable!
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/3s/6lmkv5_10g1g4vx768t1yh_0g4tfkb/T/ipykernel_50845/280325205.py in <module>
----> 1 my_string[-1] = '?' # changes are not allowed - strings are immutable!

TypeError: 'str' object does not support item assignment
```

[75]:
```
my_list = list(my_string) # Convert your string to a list to change it
```

[76]:
```
my_list[-1] = '?'
```

[77]:
```
my_modified_string = ''.join(my_list) # combine your list into a string again
my_modified_string
```

[77]:
```
'hello world?'
```

Discussion session III: Dictionary definition

[78]:
```
# create your fist dictionary
```

[82]:
```
dna_to_rna = {}
dna_to_rna['A'] = 'A'
dna_to_rna['T'] = 'U'
dna_to_rna['C'] = 'C'
dna_to_rna['G'] = 'G'
dna_to_rna
```

[82]:
```
{'A': 'A', 'T': 'U', 'C': 'C', 'G': 'G'}
```

[80]:
```
#or in one line
dna_to_rna = {'A':'A','T':'U','C':'C','G':'G'}
```

Discussion session III: Dictionary usage

[117]:
```
# dictionary operations
```

[118]:
```
dna_to_rna['U'] # access an element with its key
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/var/folders/3s/6lmkv5_10g1g4vx768t1yh_0g4tfkb/T/ipykernel_50845/3658080417.py in <module>
----> 1 dna_to_rna['U'] # access an element with its key

KeyError: 'U'
```

[119]:
```
'U' in dna_to_rna # Test if a key is in the dictionary
```

[119]:
```
False
```

[120]:
```
dna_to_rna.keys() # see all keys in the dictionary
```

[120]:
```
dict_keys(['A', 'T', 'C', 'G'])
```

[121]:
```
dna_to_rna + dna_to_rna # Dictionaries don't support concatenation
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/3s/6lmkv5_10g1g4vx768t1yh_0g4tfkb/T/ipykernel_50845/1693159149.py in <module>
----> 1 dna_to_rna + dna_to_rna # Dictionaries don't support concatenation

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

Functions

- function: stores instructions (not values)
- basic use: function_name(arguments)
- many functions loaded by default in python, e.g.
  - str() - convert an object into a string
  - int() - convert an object into an integer
  - float() - convert a object into a floating point number
  - type() - tells you the type of an object
- see many others in the cheat-sheet and find them on https://docs.python.org/3/library/functions.html

Function example

[122]:
```
# Functions
```

[123]:
```
# let's create a function using def (for define)
def add_things(a, b):
    result = a+b
    return result
```

[124]:
```
add_things(3,5)
```

[124]:
```
8
```

[125]:
```
add_things('Hello ', 'there')
```

[125]:
```
'Hello there'
```

[127]:
```
add_things('3',5) # reminder: we need to have the same type to add elements
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/3s/6lmkv5_10g1g4vx768t1yh_0g4tfkb/T/ipykernel_50845/3573978534.py in <module>
----> 1 add_things('3',5) # reminder: we need to have the same type to add elements

/var/folders/3s/6lmkv5_10g1g4vx768t1yh_0g4tfkb/T/ipykernel_50845/2105753661.py in add_things(a,
      1 # let's create a function using def (for define)
      2 def add_things(a, b):
----> 3     result = a+b
      4     return result

TypeError: can only concatenate str (not "int") to str
```

Methods

- functions of a specific object class
- access by <variable_name>.<method>(*arguments*)

```
>>> s = 'The quick brown fox jumps over the lazy dog'
>>> s.split()
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
>>> s.split('fox')
['The quick brown ', ' jumps over the lazy dog']
```

- in iPython: write . after a variable name and press <tab> to get an overview about available methods
- alternatively: dir(object)
- use ? / ?? (in ipython) or help() to get information about a method and its arguments e.g. ?my_str.split or help(my_str.split)
- alternatively: google or ask chatGTP

Break

Part 2:
Writing Python code

Why write a script?

- Organize your commands in a text file and build more complicated workflows that can be executed at once.
- save typing, make your work reproducible

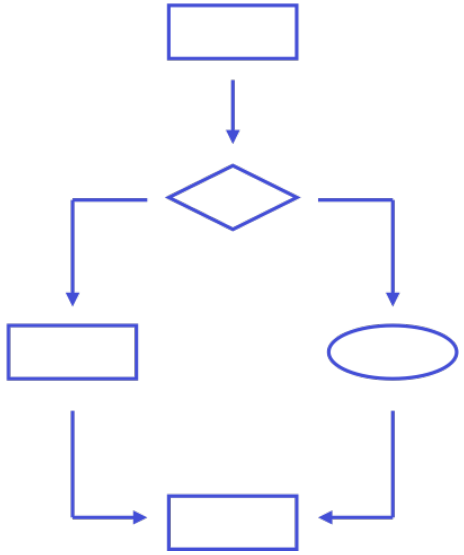
- Run it at any point by executing your script
- Use comments (#) to describe your code
- use the print() function to print the output or an intermediate result

```
>>> print("Hello","Python!")
Hello Python!
```

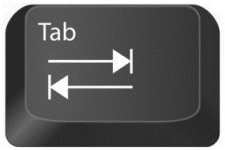
Conditional statements

- Use if/else clauses to make decisions

```
if boolean_expression:
    print("The statement is True")
else:
    print("The statement is False")
```



- Remember the indentation!
  - ➢ While other programming languages use brackets or end statements, Python uses whitespace to structure code.
  - ➢ Simply use tabulator to indent.



Loops

- Loops are essential for repeating an action several times
- The **for** loop executes the nested statements as many times as there are elements in the input list.

```
input_list = [1,2,3,4,5]
for my_number in input_list:
    print(my_number)
```

```
1
2
3
4
5
```

for loop for reading text files

```
f = open('my_file.txt','r')
for line in f:
    print(line)
f.close()
```

```
1st line

2nd line

3rd line
```

File: my_file.txt

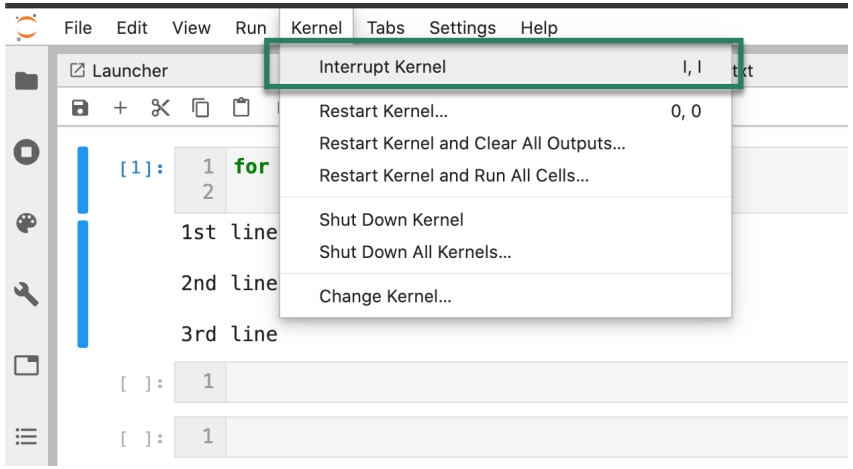
1st line
2nd line
3rd line
EOF

Conditional loop

Use the **while** loop to continue an action until a condition is not satisfied anymore

```
>>> number_apples = 3
>>> while number_apples > 0:
...     number_apples -= 1
...     print("Ate an apple,",number_apples,'left')
...
Ate an apple,  2 left
Ate an apple,  1 left
Ate an apple,  0 left
```

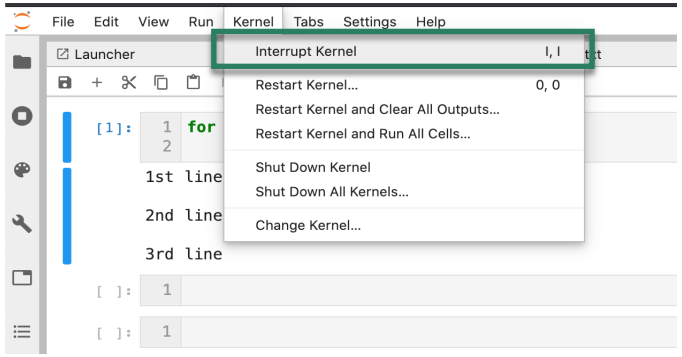
in case you need to terminate your loop…



Session IV: Loops and friends

```
# 1. try out this for loop
for my_number in range(6):
    print(my_number)

# 2. A new loop key-word: continue
for my_number in range(6):
    if my_number == 3:
        continue
    print(my_number)
print('after loop: ', my_number)

# 3. try using break instead of continue

# 4. when will this while loop finish?
my_number = 1
from time import sleep
while my_number < 5:
    print("hurray! my number is increasing:", my_number)
    sleep(1)
```

in case you need to terminate your loop…



Discussion session IV: Loops and friends

[129]:
```
# 1. try out this for loop
for my_number in range(6):
    print(my_number)
```

```
0
1
2
3
4
5
```

Discussion session IV: Loops and friends

[141]:
```
# 2. A new loop key-word: continue
for my_number in range(6):
    if my_number == 3:
        continue # skips the current iteration
    print(my_number)
print('after loop: ', my_number) # loop continued until the end
```

```
0
1
2
4
5
after loop:  5
```

[142]:
```
# 3. A new loop key-word: continue
for my_number in range(6):
    if my_number == 3:
        break # terminates the loop immediately
    print(my_number)
print('after loop: ',my_number) # loop has terminated prematurely
```

```
0
1
2
after loop:  3
```

Discussion session IV: Loops and friends

[144]:
```
# 4. when will this while loop finish?
my_number = 1
from time import sleep
while my_number < 5:
    print("hurray! my number is increasing:", my_number)
    # my_number += 1 # don't forget to update your counter!!
    sleep(1)
```

```
hurray! my number is increasing: 1
hurray! my number is increasing: 1
hurray! my number is increasing: 1
hurray! my number is increasing: 1
hurray! my number is increasing: 1
hurray! my number is increasing: 1
hurray! my number is increasing: 1
hurray! my number is increasing: 1
```

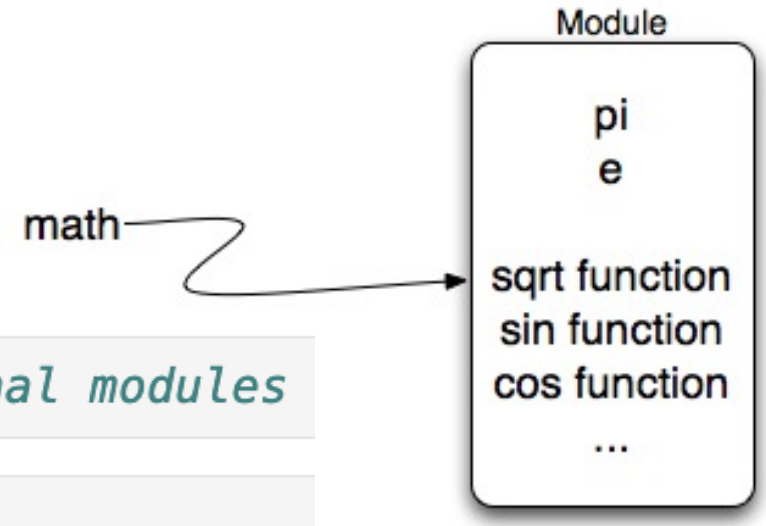
```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/var/folders/3s/6lmkv5_10g1g4vx768t1yh_0g4tfkb/T/ipykernel_50845/3886309862.py in <module>
      5     print("hurray! my number is increasing:", my_number)
      6     # my_number += 1 # don't forget to update your counter!!
----> 7     sleep(1)

KeyboardInterrupt:
```

python modules

Import statement

- Use **import** to load additional modules for more functionalities

- For example the **math** module:




```
[6]: # import statement to load additional modules
```

```
[7]: import math
     print('Pi is equal to', math.pi)
     print('or in degrees', math.degrees(math.pi))
```

```
Pi is equal to 3.141592653589793
or in degrees 180.0
```

more details at: https://docs.python.org/3/library/math.html

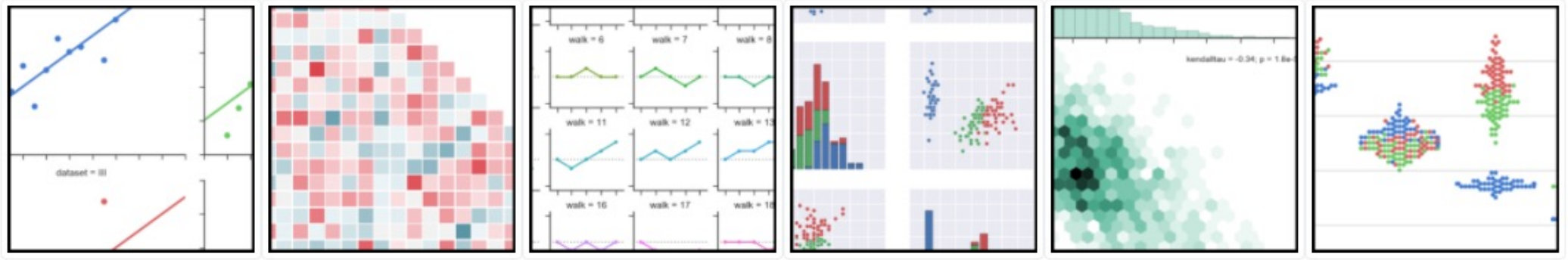
Much more to discover

- ipython magics make your life a lot easier
  - time your code, debug it, call other languages
- make python lightning fast using numpy, scipy, and cython/pypy/numba
- visualize your data with seaborn, bokeh and matplotlib

IP[y]: IPython

NumPy

Cython



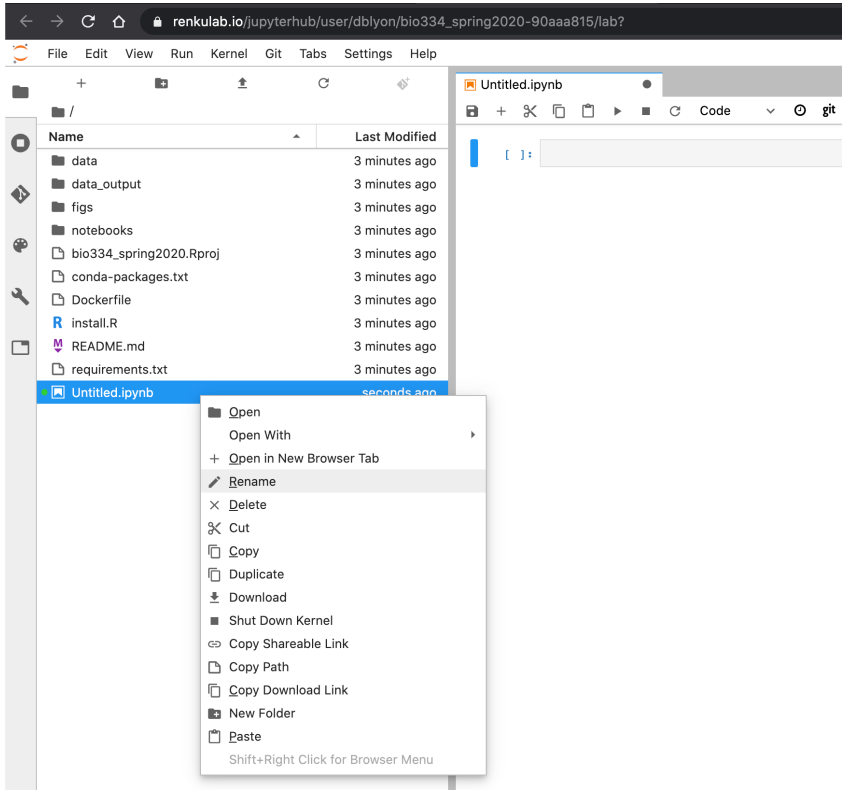
Further reading

- https://snakify.org/ (interactive tutorial)
- http://www.diveintopython.net/(comprehensive, general purpose)
- http://swcarpentry.github.io/python-novice-inflammation/ (scientific python by example)
- http://www.scipy-lectures.org/intro/intro.html (intro by the scientific python consortium)
- https://github.com/dblyon/pandasintro (extensive introduction to pandas)
- https://cs50.harvard.edu/college/2021/spring/weeks/0/

Exercise session

- Create a new Python3 Jupyter Notebook, save it to the folder */Bio334/02_python_introduction/exercises* and start exploring

- Change the name of your newly created file by right clicking on *Untitled.ipynb*
- To execute a cell press Shift + Enter



optional exercise slides follow

Optional Exercise #3

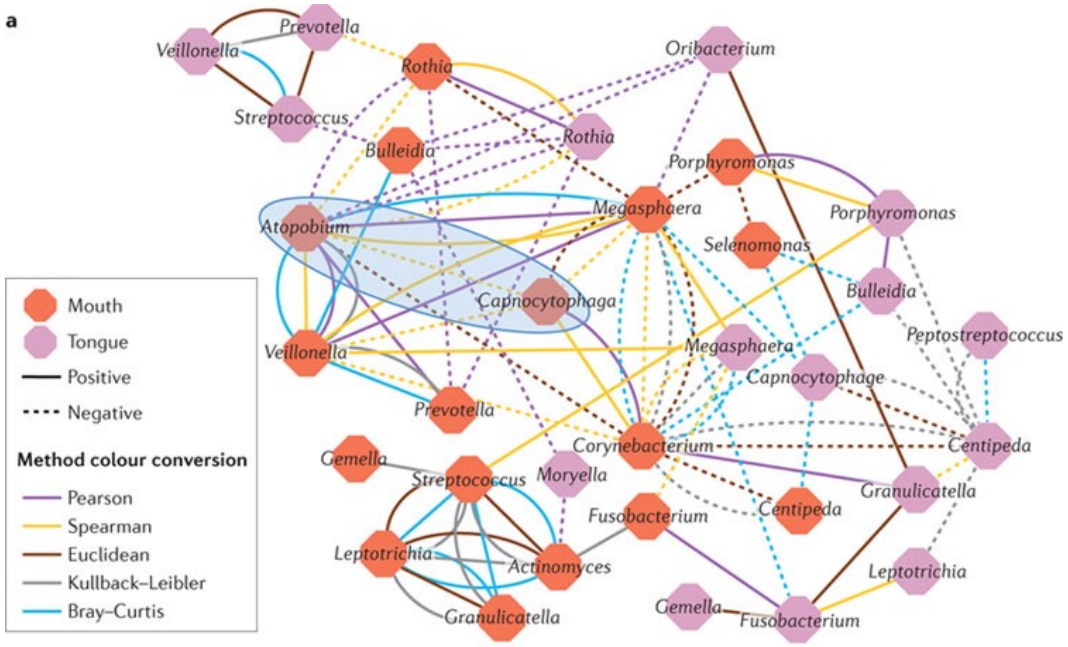
Learn the basics of **Pandas**, a powerful python module for data analysis, including reading, writing, filtering, merging, arithmetic operations on and sorting of tabular data

```
age animal  priority    visits
b   3.0 cat yes 3
d   NaN dog yes 3
f   2.0 cat no  3
```

	age	animal	priority	visits
b	3.0	cat	yes	3
d	NaN	dog	yes	3
f	2.0	cat	no	3

Optional Exercise #4

Create a pipeline to infer simple ecological relationships in the Human Microbiome



Optional exercise session

- Create a new python file in JupyterLab
- To execute, launch a terminal tab and enter
  - ipython <script_name>.py

- Good luck!

sys module

- A built-in module that contains system-specific parameters and functions
- For example we can use it to read arguments from the command line:

```
Script: say_hello.py
import sys
print('Hello there', sys.argv[1])
   # Command line arguments are in sys.argv[1], sys.argv[2]...
   # sys.argv[0] is the script name itself
```

```
davide$ python say_hello.py bio334_Students
Hello there bio334_Students
```

os module

- Another built-in module that contains miscellaneous operating system interfaces
- For example we can use it to obtain the name of the user currently logged in:

Script: check_directory.py

```
1  import os
2
3  dir_ = "Bio334"
4  if not os.path.exists(dir_):
5      print("It seems you haven't cloned the git repository yet. \
6            Please enter the following command in a terminal window.")
7      print("git clone https://github.com/meringlab/Bio334.git")
8  else:
9      print("Great! The 'Bio334' directory exists. You are good to go.")
```

Run the script in a terminal window

```
base > work > bio334_spring2020_dbl > master > 3? > $ python check_directory.py
Great! The 'Bio334' directory exists. You are good to go._
```