Introduction to Python

BIO334

Maria Heimlicher & Mihai Danaila

Christian von Mering's group

courtesy of Sumner, Lyon, Tackmann, Dmitrijeva & Gable

maria.heimlicher@mls.uzh.ch

mihai-cosmin.danaila@uzh.ch

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

What is Python?

1. Dynamic, interpreted **programming language**
2. **Simple syntax** with fast learning curve

Most Loved, Dreaded, and Wanted Languages

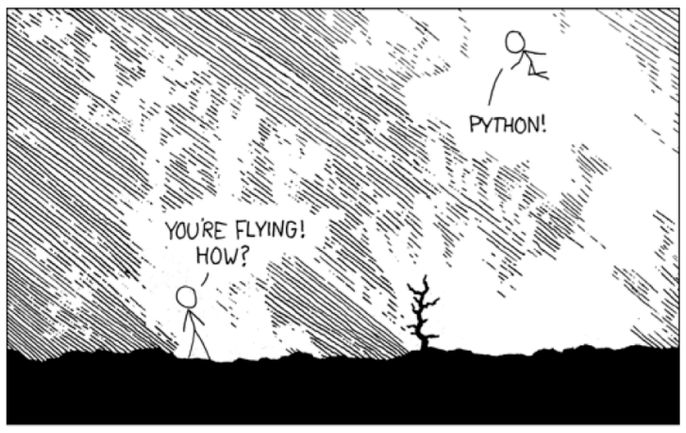
Loved | Dreaded | Wanted

Rust	83.5%	
Python	73.1%	
TypeScript	73.1%	
Kotlin	72.6%	
WebAssembly	69.5%	

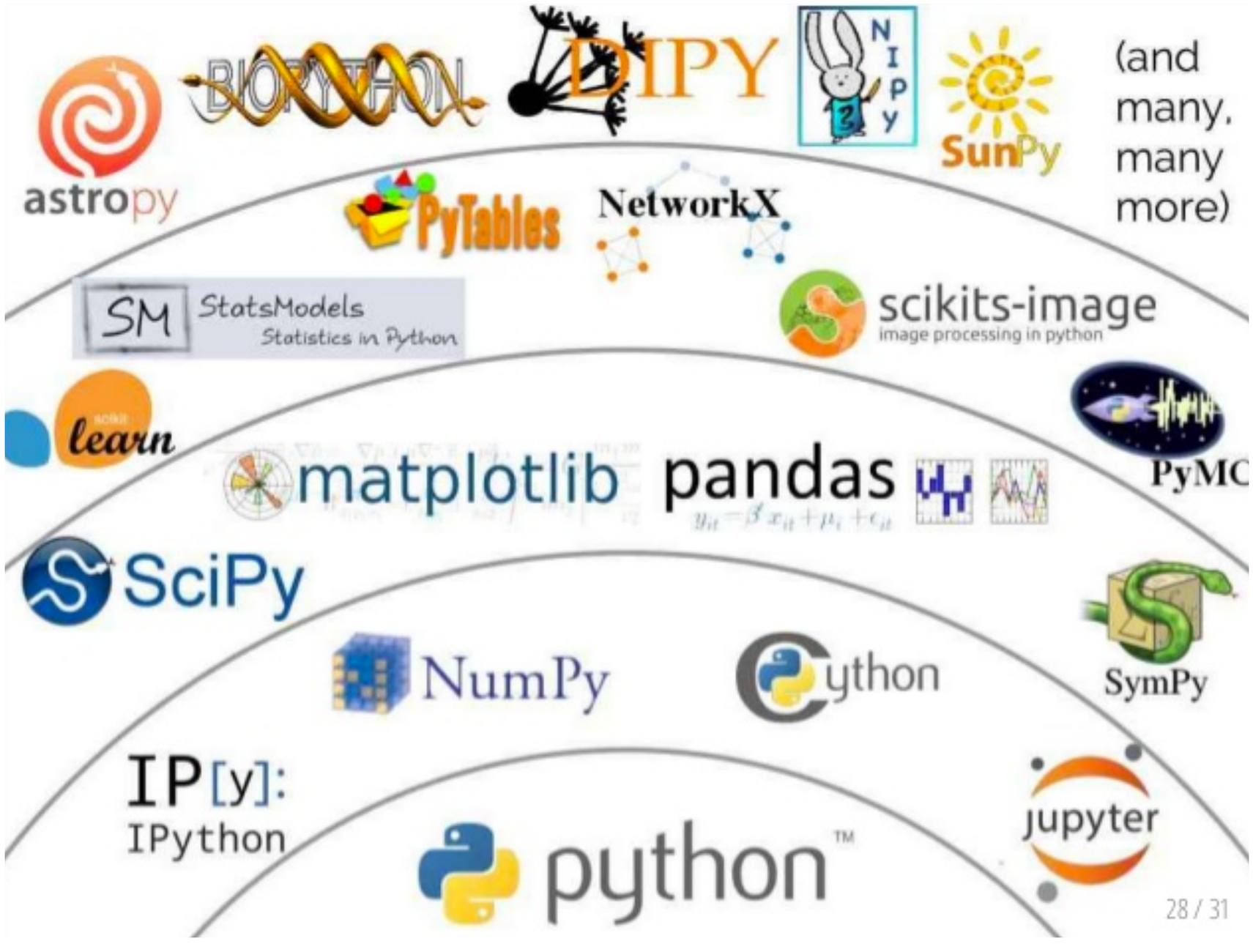
Most Loved, Dreaded, and Wanted Languages

Loved | Dreaded | Wanted

Python	25.7%	
JavaScript	17.8%	
Go	15.0%	
TypeScript	14.6%	
Kotlin	11.1%	



Scientific Python stack



Today's program

1. Introduction to programming
   basic concepts with small hands-on sessions in JupyterLab using iPython
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 4.15pm

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:
  - use descriptive names, NO spaces, don’t start with numbers

Operators

1. Arithmetic operations
   - addition + , subtraction –
   - division / , multiplication *
   - exponent **
   - parenthesis ( )

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

2. Boolean operations
   - return `True` or `False`
   - == , > , < ,
   - & (AND) , | (OR)

```
>>> 1 < 2
True
>>> 3 > 34
False
>>> 23 == 45
False
>>> 34 != 323
True
```

First live session

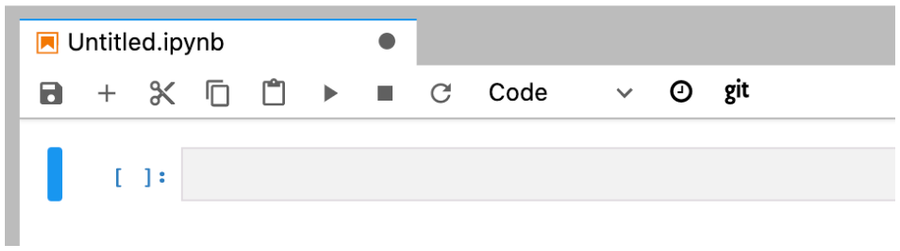
Show how to use **JupyterLab**
and
checkout **Cheat Sheets**

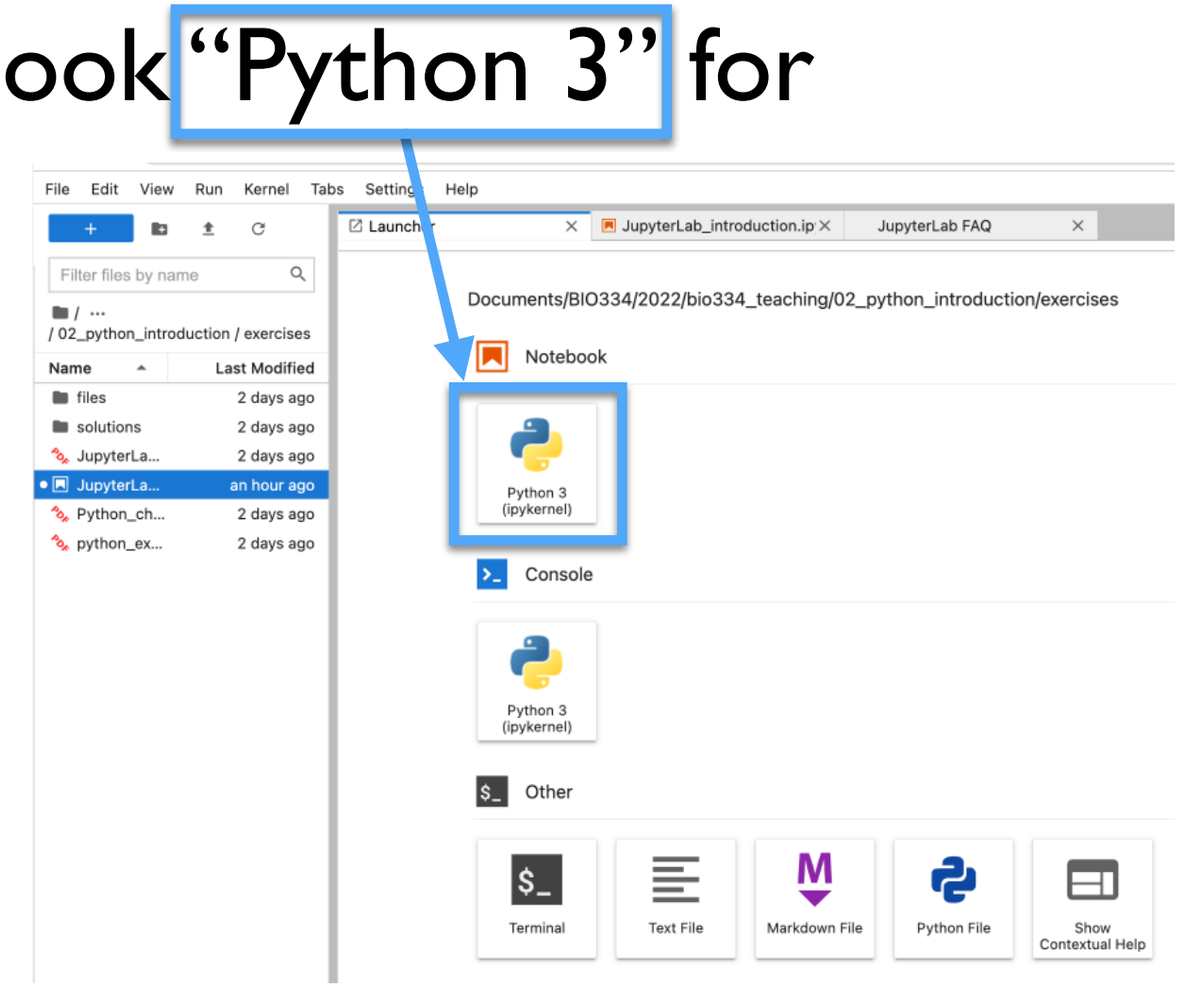
To access the files you need, enter the following command in a terminal window

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

First live session

- Open the Anaconda Navigator from the Applications and launch Jupyter Lab
- Launch Jupyter Notebook "Python 3" for an interactive session





Session 1:
Type a command and hit `shift + enter`

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

Arithmetics

```
>>> welcome_message = 'hello world!' # hit [shift + enter]
>>> welcome_message
'hello world!'

# Use python as your advanced calculator
>>> a = 4
>>> b = a + 3
>>> (a + b) * 4
44

# float division (caveat: Python 2 would return 0!)
>>> a / 8
0.5

>>> a**2
16
```

Concatenation

```
# Let's try with strings

# use + to add two strings
>>> welcome_message + welcome_message
'hello world!hello world!'

>>> a = '4'

# We need to have the same type to add elements
>>> a + 3      # anything strange?
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects

# whats the difference?
>>> a + str(3)
'43'
>>> int(a) + 3
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 [ ]
  - e.g.

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

List data structure

- get subsets 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. sort, reverse, insert, search

Session 11: Work with lists

```
# create your first list and 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 variable types in your list

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

my_mixed_list

# apply the arithmetic operators on lists

[1,2,3] + [3,4,6]

['Hello'] * 4
```

Accessing list elements

```
# create your first list and try accessing it with indices
>>> my_list = [1,2,3,4,5]
>>> my_list[1]
2
>>> my_list[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

```
# slice your list with index ranges
>>> my_list[0:3]
[1, 2, 3]
>>> my_list[-1]
5

# Check how many elements your list contains
>>> len(my_list)
5
```

List modifications

```
# mixed lists, put different variable types in your list
>>> my_mixed_list = ['UZH','founded in',1834]
>>> my_mixed_list[2] = 1834

>>> my_mixed_list
['UZH', 'founded in',1834]

# delete an element at a specified index
>>> del my_mixed_list[1]

>>> my_mixed_list
['UZH', 1834]

# add another element to the list with the append command
>>> my_mixed_list.append('A.D.')

>>> my_mixed_list
['UZH', 1834, 'A.D.']
```

List concatenation

```
# apply the arithmetic operators on lists

# use + to make a longer list out of two small ones

>>> [1,2,3] + [3,4,6]
[1, 2, 3, 3, 4, 6]

# Use * to form a new list by repeating the content of a list

>>> ['Hello'] * 4
['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 = 'MRHIAHTQRCLSRL'
>>> list(my_sequence)
['M', 'R', 'H', 'I', 'A', 'H', 'T', 'Q', 'R', 'C', 'L', 'S', 'R', 'L']
```

- 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 11: 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)

# 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'

#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
```

String lists

```
# accessing a character by index

>>> my_string = 'hello world!'
>>> my_string[6]
'w'

# Changes are not allowed

>>> my_string[-1] = '?'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment

# Convert your string to a list to do that

>>> my_list = list(my_string)
>>> my_list[-1] = '?'

# Combine your list into a string with the join method
>>> my_modified_string = ''.join(my_list)
>>> my_modified_string
'hello world?'
```

Dictionary definition

```
# 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
{'A': 'A', 'C': 'C', 'T': 'U', 'G': 'G'}

#or in one line

>>> dna_to_rna = {'A':'A','T':'U','C':'C','G':'G'}
>>> dna_to_rna
{'A': 'A', 'C': 'C', 'T': 'U', 'G': ‘G'}
```

Dictionary usage

```
#access an element with its key
>>> dna_to_rna['U']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'U'

# Test if a key is in the dictionary
>>> 'U' in dna_to_rna
False

# See all keys in the dictionary
>>> list(dna_to_rna.keys())
['A', 'C', 'T', 'G']

# Dictionaries don't support concatenation
>>> dna_to_rna + dna_to_rna
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'dict' and 'dict'
```

Functions

- function: stores instructions (not values)
- basic use:
  - `function_name(arguments)`
- 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 other in the cheat-sheet and find them on
  - https://docs.python.org/3/library/functions.html

Functions

```
# Let's create a function using “def” (for define)
```

[ ]:
```
def add_things(a, b):
    result = a + b
    return result
```

```
# We need to have the same type to add elements
```

```
add_things(3, 5)
```

```
8
```

```
add_things("Hello ", "there")
```

```
'Hello there'
```

```
add_things("3", 5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-df373cddff49> in <module>
----> 1 add_things("3", 5)

<ipython-input-12-eb66cd937434> in add_things(a, b)
      1 def add_things(a, b):
----> 2     result = a + b
      3     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. `?str.split` or `help(str.split)`
- Alternatively: google!

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
  - Use comments (#) to describe your code
  - Run it at any point by executing your script

Scripting ingredients

- Use the **`print`** function to print the output

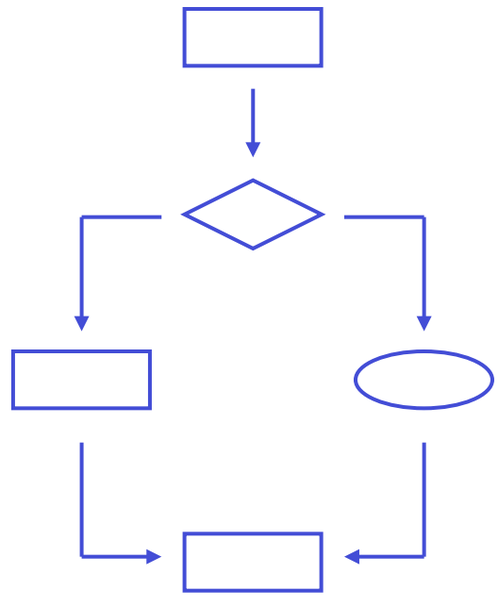
```
>>> print(“Hello”,”Python!”)
Hello Python!
```

```
print(“Hello”,”Python!”) # use this (for Python 3)
print 'hello world!'  # not this (for Python 2)
```

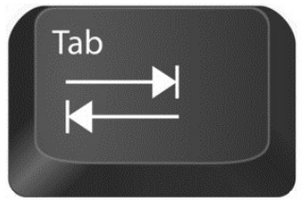
Conditional statements

- 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
```

- Note: Don't forget the colon (:) at the end.

`for` loop for reading

[1]:
```
1 for line in open("my_file.txt", "r"):
2     print(line)
```

```
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

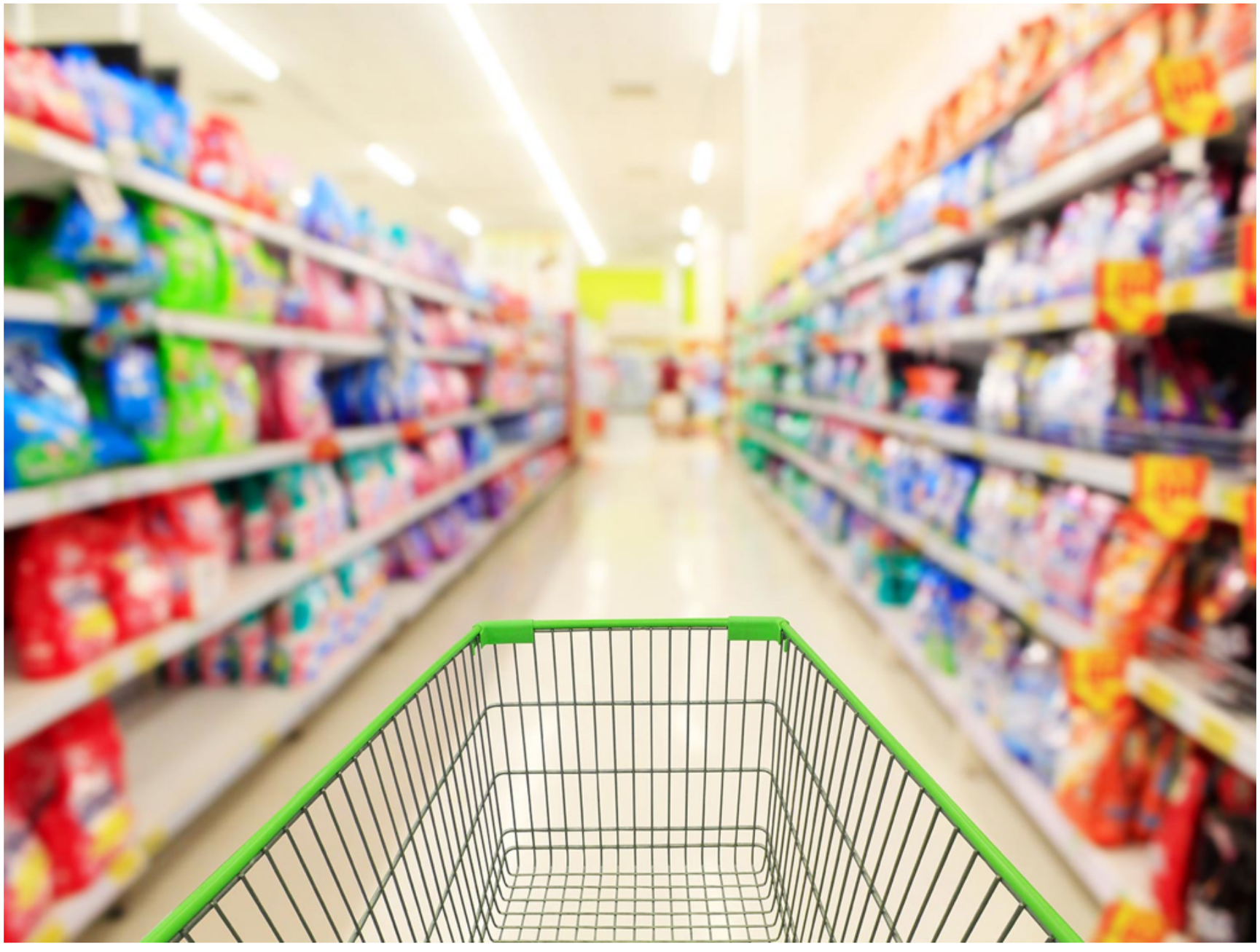
```
>>> number_apples = 3
>>> while number_apples > 0:
...     number_apples -= 1
...     print(“Ate an apple,”,number_apples,'left')
...
Ate an apple,  2 left
Ate an apple,  1 left
Ate an apple,  0 left
```

Session IV: Loops and friends

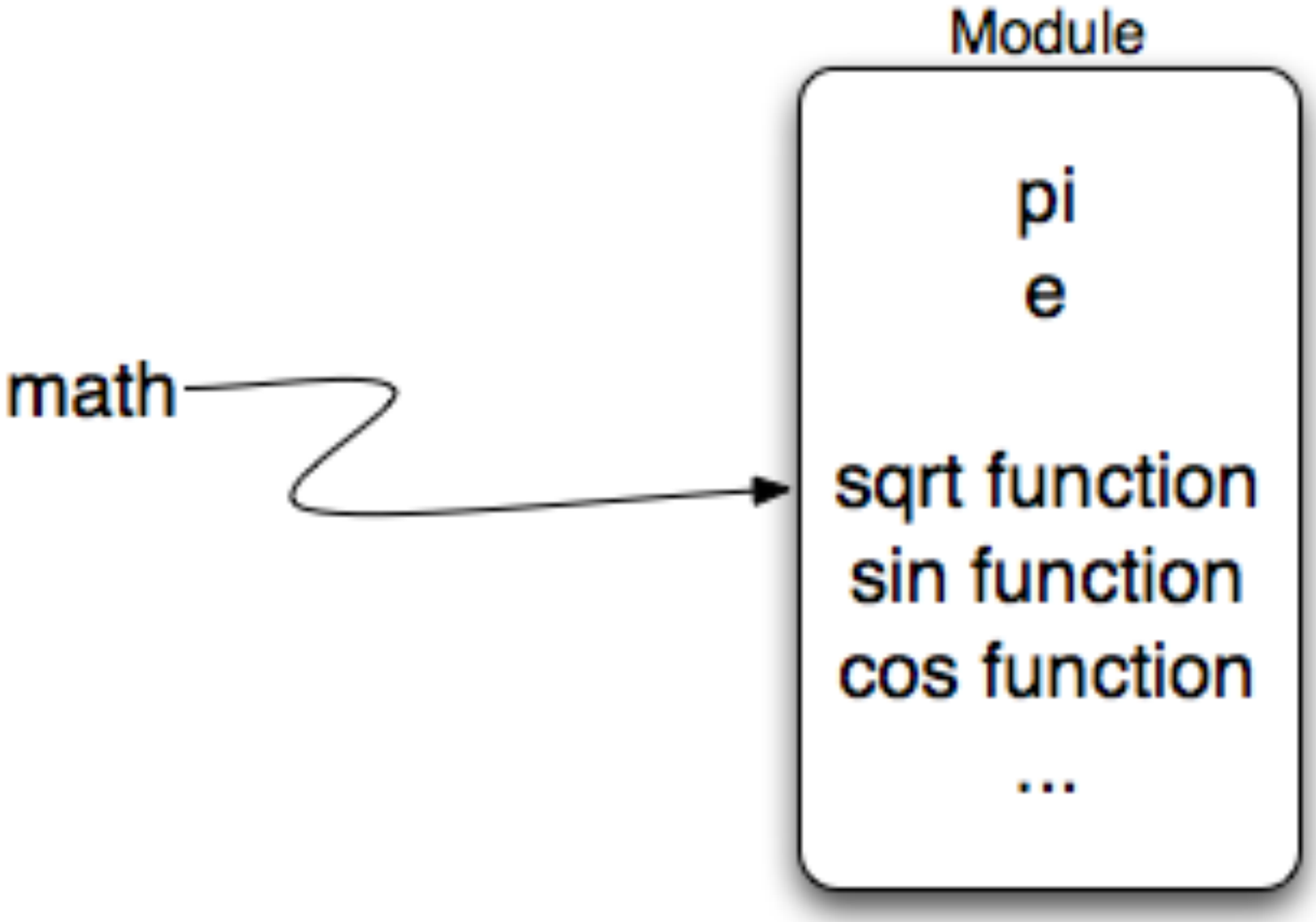
```
# 1. try out this for loop

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 == 5:
         continue
    print(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)
```

HINT: use Kernel —> Interrupt Kernel to stop the running program

python modules



`math` module



Import statement

- Use **`import`** to load additional modules for more functionalities
- For example the **`math`** module:

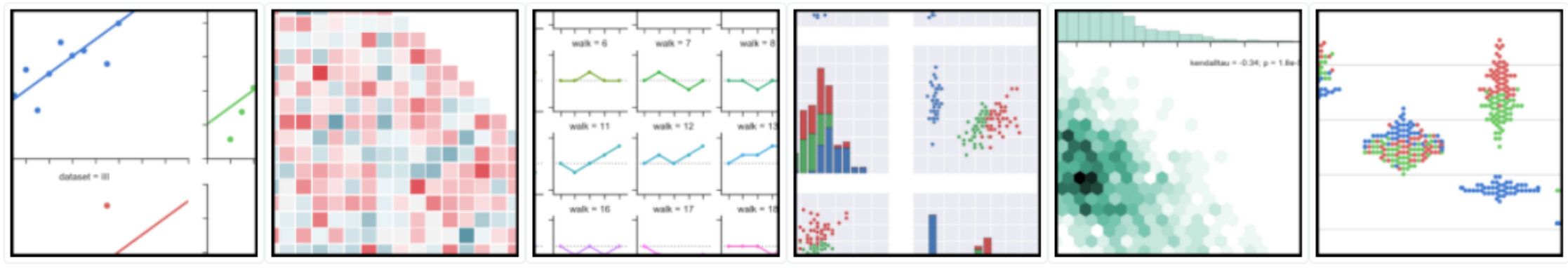
```
Script: print_pi.py

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

```
davide$ python print_pi.py
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`

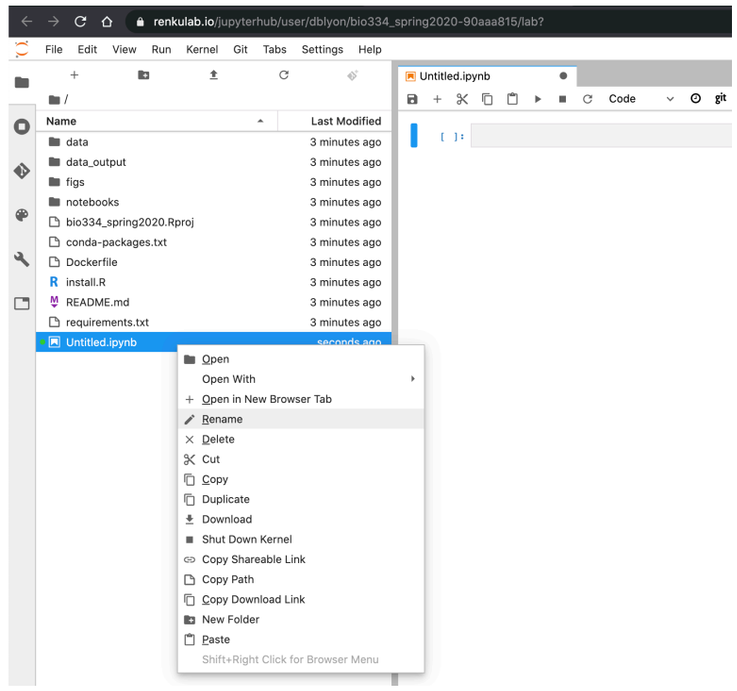


Further reading

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

Exercise session

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



2. Change the name of your newly created file by right clicking on `Untitled.ipynb`

3. To execute a cell press `Shift + Enter`

4. To access the files you need enter the following command in a terminal window

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

optional exercise slides follow

Optional Exercise #3

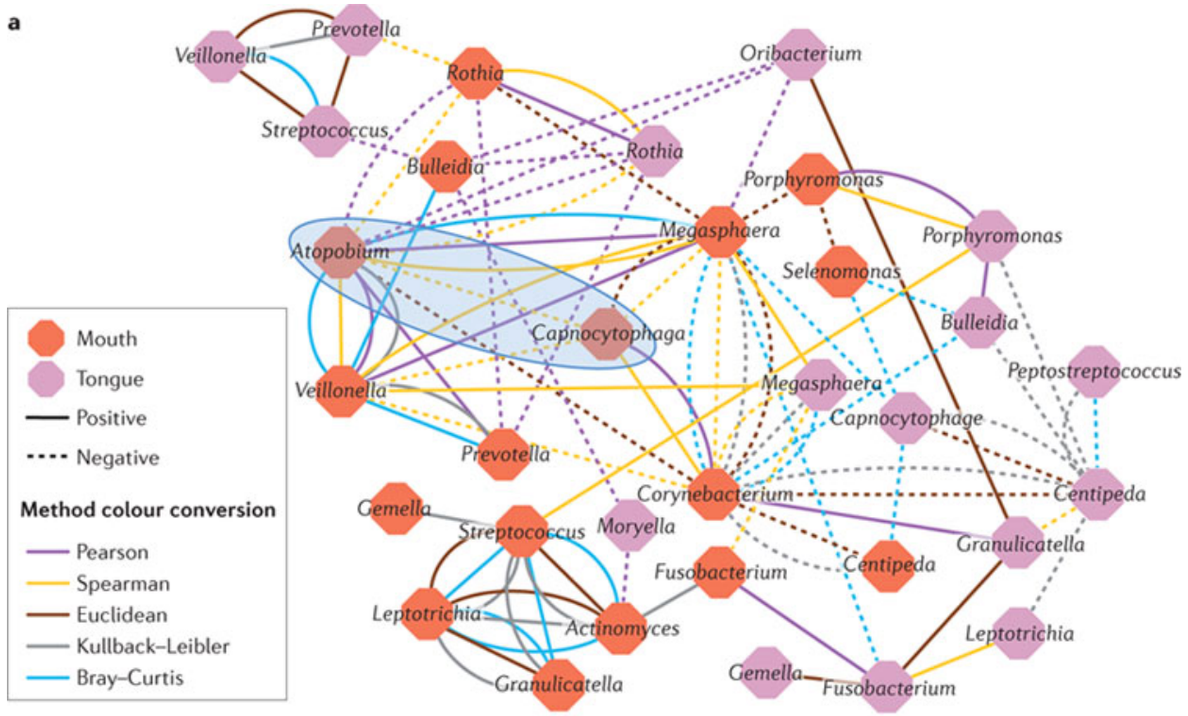
Learn the basics of **Pandas**, a powerful python module for data analysis, including reading, writing, filtering, merging, arithmetic operations on and sorting of tabular data

```
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 text document and rename it, giving it a ".py" ending
- To execute, launch a terminal tab and enter
  - `ipython <script_name>.py`
- Good luck!

`sys` module

1. A built-in module that contains system-specific parameters and functions
2. 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

1. Another built-in module that contains miscellaneous operating system interfaces
2. 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._
```