Welcome to BIO-210

Applied software engineering for life sciences

September 9th 2024 – Lecture 1

Prof. Alexander MATHIS

EPFL

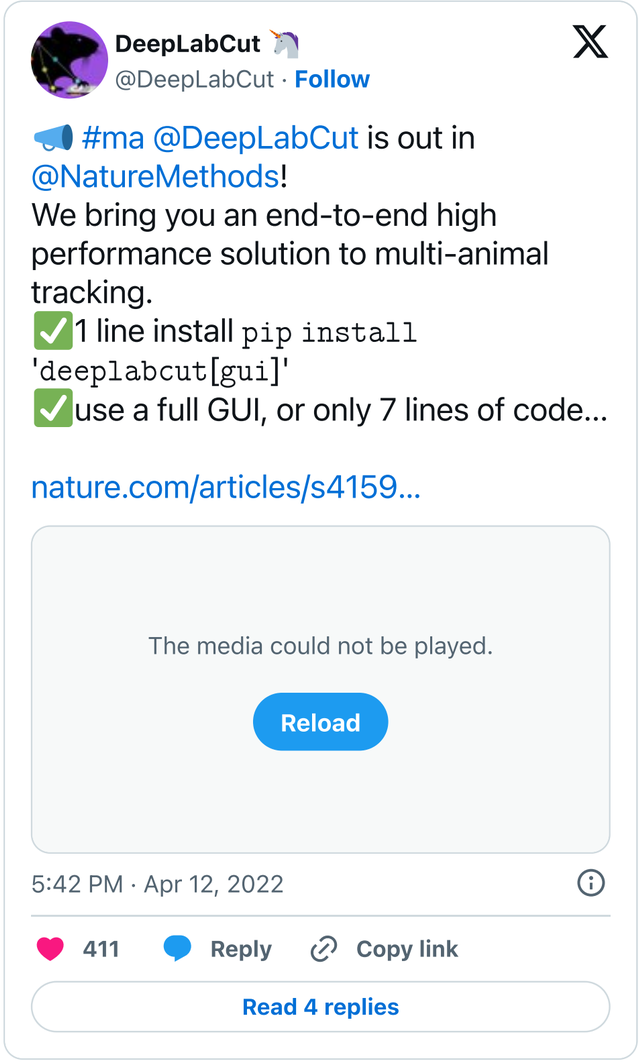
Today’s content

- Course overview
- Introduction to Python

Who am I, and what do I do with Python?

- Assistant Professor at EPFL since 8/2020
- In my lab we carry out computational neuroscience and machine learning research
- We mostly use Python --> check out our research code
- We develop open-source software, e.g.: DeepLabCut, DLC2action, AmadeusGPT, ...

NOTE: underlined terms indicate hyperlinks!



Course content

This is a project-based learning course. You will develop a software project in teams of three students.

Software engineering content:

- Python (object types, statements, functions, packages, object oriented programming)
- Distributed version control via git
- Debugging, profiling, refactoring
- Unit and functional testing
- Project and code documentation

Those topics will be covered in a learning-by doing fashion, while you expand your project. Different versions need to be released according to a schedule (see next page). Code updates should be version controlled (git).

Life science engineering content:

- *Models* from developmental biology and neuroscience

Class schedule

	Date	Topic	Software version	Software releases	Grading / Feedback
0	09/09/2024	Python introduction I			
1	16/09/2024	Public holiday			
2	23/09/2024	Python introduction II			
3	30/09/2024	Git and GitHub (+installation VS Code)			
4	07/10/2024	Project introduction	v1		
5	14/10/2024	Functionify	v2	v1	
6	21/10/2024	EPFL fall break			
7	28/10/2024	Visualization and documentation	v3	v2	code review (API)
8	04/11/2024	Unit-tests, functional tests	v4	v3	
9	11/11/2024	Code refactoring	v5	v4	graded (tests)
10	18/11/2024	Profiling and code optimization	v6	v5	code review
11	25/11/2024	Object oriented programming	v7	v6	graded (speed)
12	02/12/2024	Model analysis and project report	v8	v7	code review (OO)
13	09/12/2024	Work on project			

Our teaching team

Exercises:

- Teaching assistants: Mu Zhou, Albert Dominguez Mantes, Seda Radoykova, Shaokai Ye, Haozhe Qi, Oliver Ulrich, Andy Bonnetto
- Student assistants: Huyen Nguyen, Jennifer Shan, Jeremy Barghorn, Leo Ganser, Leonardo Tredici, Louise Montlahuc, Lucie Manson, Maylis Muller, Pires Joana, Benjamin Gabriel Mancini, Eva Quinto, Ismael Salioski, Wesley Monteith, Viva Berlenghi, Henryk Viana, Timothe Dufour

Contacting me:

- Email: alexander.mathis@epfl.ch
- Office hours at SV 2811, Monday 15:15 - 16

Logistics

- Monday 10 - 12: lecture in CE12
- Monday 13 - 15: exercises (5 groups)
  - CO4 ← A-B + Y + Z
  - CO5 ← C-F+V
  - CO260 ← G-L+W
  - CO6 ← M-P
  - CO023 ← Q - U
- Monday 15:15 - 16: my office hours at SV 2811

Online:

- Moodle – announcements, quizzes, and slides (posted before class)
- Ed – forum for questions
- GitHub – we share code there

Assessment

Final grade comprises three independent scores:

- 40% individual assessments (via Moodle)
  - several quizzes (see next slide)
- 35% evaluation of the group project (tests, profiling and project report)
  - graded versions v4, v6 and v8
- 25% individual contributions to the group project (participation and individual contributions to code)
  - we check code contributions on GitHub
  - participation assessed in exercise sessions

See Fiche de cours.

Quizzes

- Week 3 - 7.5% of the grade (online available for 1 work week).
- Week 5 - 12.5% of the grade. This will be *in person* for 20 min at the beginning of the exercises.
- Week 7 - 12.5% of the grade. This will be *in person* for 20 min at the beginning of the exercises.
- Week 9 - 12.5% of the grade. This will be *in person* for 20 min at the beginning of the exercises.
- Week 11 - 7.5% of the grade (online available for 1 work week).

Note for the in-person quizzes, your score will be based on the best two.

Questions?

What is Python?

a general-purpose programming language

Why Python?

Pros:

- software quality: highly readable code
- *hence* reusable and maintainable
- developer productivity: code is shorter (about 1/5 of C++/Java), and runs immediately
- large support libraries (large standard library, Numpy, etc.)
- dynamic typing (no declarations)
- automatic memory management
- it’s free and open source

Cons:

- speed, but … numpy, TensorFlow, PyTorch, cython…

Why Python?

- simple programming structure
- flexible and easy to learn
- vibrant community python.org, stackoverflow
- many available packages, e.g. via Python Package Index (PyPI), Conda (an open source package management system)
- excellent Application Programming Interface (APIs) to the open-source machine learning libraries TensorFlow, PyTorch, etc.
- great to learn as a second language (after you learned C++)

C++ code

```cpp
#include <iostream>
int main() {

    std::cout << "Hello, World!";

    return 0;

}
```

Python code

```python
print('Hello, world!')
```

Notes:

- shorter code
- no line determination ";"
- no header files
- no return value needed

Quiz: What does this code do?

```cpp
#include <iostream>

int main() {
  int number;

  std::cout << "Enter an integer: ";
  std::cin >> number;

  if (number > 0) {
    std::cout << "You entered a positive integer: " << number << std::endl;
  }
  else {
    int number2 = 3;
  }
  std::cout << "Program ended.";
  return 0;
}
```

C++ code

```cpp
#include <iostream>

int main() {
  int number;      // declare variable /w type

  std::cout << "Enter an integer: ";
  std::cin >> number;

  if (number > 0) {
    // Print number if it is positive.
    // Also notice allowed line breaks
    std::cout << "You entered"
    "a positive integer: "
    << number << std::endl;
  }
  else {
    // Let's just assign a new number
    int number2 = 3;
  }
  std::cout << "Program ended.";
  return 0;
}
```

Python code

```python
number = int(input('Enter an integer: '))

if number> 0:
  # Print number if it is positive.
  # Also notice allowed line breaks
  print("You entered a \
  positive integer ", number)
else:
  # Let's just assign a new number
  number2=3   # type not defined

print("Program ended.")
```

Notice:

- lack of {,}
- indenting in `if`/`else`
- Comments with `#`

Who uses Python?

- Google
- Dropbox
- Spotify
- Facebook/Meta
- …
- scientific research
  - many labs at *EPFL* (mine included)
  - case study: first image of a black hole!

Google decided:

“Python where we can, C++ where we must.”

Sources

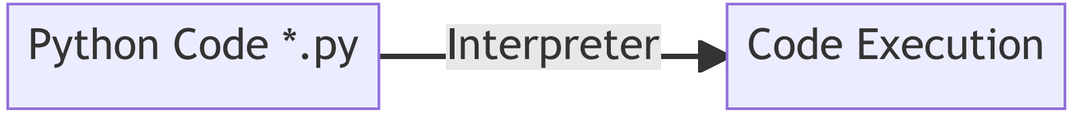
Image Credits: Event Horizon Telescope Collaboration

What can you do with Python?

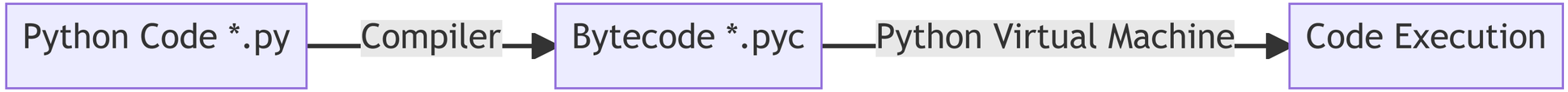
Pretty much anything…

- scientific computing (many powerful libraries exist: scipy, numpy, scikit-learn,…)
- AI, machine learning, computer vision, NLP, … (Libraries: TensorFlow, PyTorch, HuggingFace, JAX, …)
- data visualization (e.g., Matplotlib, Bokeh, etc.)
- graphical user interfaces (GUI) (e.g. overview, wxPython)
- web frameworks and cloud stacks
- rapid prototyping
- microcontroller development
- data analytics and distributed processing
- system-level glue code
- …

Python is an interpreted language



More precisely CPython, the reference implementation of the Python language, is a bytecode interpreter. When executing a script (*.py file), a compilation step generates bytecode (a *.pyc file/s), that is then interpreted and executed by a virtual machine.



In contrast, for C++:

Quick demo …

- showing the interactive Python console
- showing a Jupyter notebook
- in today's exercises, you'll use EPFL's internal Jupyter Hub --> https://noto.epfl.ch/

Learning by trying it out

```
slidev — IPython: alex/slidev — ipython — 123×29
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev % ipython
Python 3.9.12 (main, Apr  5 2022, 01:52:34)
Type 'copyright', 'credits' or 'license' for more information
IPython 8.15.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: 2*4
Out[1]: 8

In [2]: 3**4
Out[2]: 81

In [3]: 33*"exciting"+"python"
Out[3]: 'excitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingex
citingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcit
ingexcitingexcitingexcitingpython'

In [4]:
```

Ipython

Questions?

Python's conceptual hierarchy

1. Programs are composed of modules
2. Modules contain statements
3. Statements contain expressions
4. Expressions create and process objects

Today we will focus on the lowest level (expressions and certain built-in objects)

Python’s core built-in objects

Object type:	Examples:
Numbers	123, 3.14, math.pi, …
Strings	'abc', 'EPFL', "Geneva", …
Lists	[1, [2, 'troi'],4], list(range(99)
Dictionaries	{'Apples': 200, 'Pears': 123.5}, dict(hours=10)
Tuples	(x,y,z), (1, [2, 'troi'],4)
Sets	set('abc'), {'E','P','F','L'}
Other core types	Booleans, types, None
Files	open('data.txt'), open(r('/home/alex/abc.bin'),'wb')
Program unit types	Functions, modules, classes

Numbers and operations

Python's core objects include integers, floating-point, complex numbers, etc.

```
>>> 3+12455       # Here and later, ```>>>```, denotes what you type in a Python shell
12458             # here is the output!
#  back to numbers:
>>> 3+12455       # integer addition
12458
>>> 1.5*4         # floating-point multiplication (note: one integer!)
6.0
>>> 2**99         # 2 to the power of 99; also pow(2,99)
633825300114114700748351602688
>>> 3-2           # subtraction
1
>>> abs(-19.2)    # absolute value
19.2
>>> 13 % 3        # Remainder of 13/3
1
```

Numbers in Python have `types`

Types of numbers:

- `int` represents integers, e.g. `2`, `14213555`
- `float` represents reals, e.g. `pi`, `1.0`, `0.000001`
- `complex` represents complex numbeers, e.g. real + imaginary
- `bool` represents Boolean values, `True` and `False`
- `type()` returns the type of Python objects

```
>>> type(5)             # again, ```>>>```m denotes what you type in a Python shell
int                     # here is the output!
>>> type(3.0)
float
>>> type(True)
bool
>>> complex(1,1)        # define a complex number
(1+1j)                  # evaluates to this!
>>> complex(0,1)**2     # remember: sqrt(-1) = j
(-1,0j)
>>> type(complex(1,1))
complex
```

Type conversion (casting)

Allows converting the type of one object to another.

For instance,

```
>>> float(5)    # converts integer to float
5.0
>>> int(4.9)    # rounds float to integer!
4

# The type can also change during a computation
>>> 12/3          # division --> note the type changes from int to float
4.
>>> 12//3         # floored quotient (division)
4
>>> 1//2
0
>>> -1//2         # floored (rounded towards -infinity)
-1
```

Simple numerical example

```
from math import pi      # module import, we will cover this later, just importing pi
radius = 3               # names and assignment statement

area = pi * radius**2    # calculating the area of a circle.
circumference = 2*pi*radius
```

Note: it is good practice to use meaningful names.

Note: radius is an integer, but area will be float (type conversion).

Brief interlude on variable names and keywords

Valid variable names:

- must begin with a letter
- are arbitrarily long sequences of letters ('a'...'z', 'A', ...,'Z'), underscore ('_') and digits that are not keywords.

For instance, those don't work:

```
>>> 1969EPFL = "founding date school name"
SyntaxError: invalid syntax
>>> 10k$ = 10000
SyntaxError: invalid syntax
>>> if = 1000                          # this is a keyword
SyntaxError: invalid syntax
```

Other keywords: `else`, `elif`, `class`, ... (we'll learn more about them later)

Quiz: What happens here?

```
>>> from math import e, pi
>>> e**(pi*complex(0,1))+complex(1,0)
1.2246467991473532e-16j
```

Quiz

Euler's identity

```
>>> from math import e, pi
>>> e**(pi*complex(0,1))+complex(1,0)
1.2246467991473532e-16j
```

This is Euler's identity: $e^{\pi * 1j} + 1 = 0$

Why is it not zero?

- Check out What every computer scientist should know about floating-point arithmetic by David Goldberg. But in essence, "Squeezing infinitely many real numbers into a finite number of bits" implies rounding and computing errors!

Questions?

Strings

- strings are concatenations of letters, special characters, numbers, and spaces
- strings are case sensitive
- strings can be *defined* by enclosing in quotation marks (") or single quotes (').

Examples:

```
>>> S = 'Geneva'        # make a 6-character string and assign it to a name
>>> S = "Lausanne"      # make a 8-character string and assign it to a name
>>> S = str(3)          # cast integer 3 to string and assign to name
>>> type(S)
str
```

Sequence operations (on strings)

```
>>> S = 'Geneva'        # make a 6-character string and assign it to a name
>>> S
'Geneva'
>>> len(S)              # Length
6
>>> S[0]                # The first item in S.
'G'
>>> S[1]                # The second item in S
'e'
>>> S[-1]               # The last item of S
'a'
>>> S[len(S)-1]         # The last item of S, the hard way
'a'
>>> S[-2]               # The penultimate item of S
'v'
```

Note: in Python indices start from 0, the second index is 1, etc. That's why we also enumerate our classes in this way.

Sequence operations (on strings)

```
>>> S = 'Geneva'        # make a 6-character string and assign it to a name
>>> S
'Geneva'
>>> len(S)              # Length
6
>>> S[0]                # The first item in S.
'G'
>>> S[1]                # The second item in S
'e'
>>> S[-1]               # The last item of S
'a'
>>> S[len(S)-1]         # The last item of S, the hard way
'a'
>>> S[-2]               # The penultimate item of S
'v'
```

Note: in Python indices start from 0, the second index is 1, etc. That's why we also enumerate our classes in this way.

More sequence operations (on strings)

```
>>> S = 'Geneva'        # make a 6-character string and assign it to a name
>>> S[1:]               # characters past the first (1:len(S))
'eneva'
>>> S                   # notice S has not changed!
'Geneva'
>>> S[1:3]              # characters from first to third
'en'
>>> S[:-1]              # everything but last item
'Genev'
>>> S+', GE'            # concatenation
'Geneva, GE'
>>> S*3                 # repetition
'GenevaGenevaGeneva'
>>> S                   # notice, S is unchanged!
'Geneva'
```

Note: `X[I:J]` means everything in `X` from offset `I` up to (but excluding) `J`.

Immutability

- strings are immutable in Python (i.e., they cannot change after they have been created)
- numbers are also immutable (obviously)

```
>>> S = 'Geneva'
>>> S[0]=S                  # immutable objects cannot be changed
... error ...
TypeError: 'str' object does not support item assignment
>>> S = S[:3]+'etics'       # but we can re-assign it! (expression to make new object)
>>> print(S)
'Genetics'
```

Immutability

- strings are immutable in Python (i.e., they cannot change after they have been created)
- numbers are also immutable (obviously)

```
>>> S = 'Geneva'
>>> S[0]=S                  # immutable objects cannot be changed
... error ...
TypeError: 'str' object does not support item assignment
>>> S = S[:3]+'etics'       # but we can re-assign it! (expression to make new object)
>>> print(S)
'Genetics'
```

Swiss greeetings and polymorphism

```
greeting1 = 'Bonjour'      # definition with single quote
greeting2 = "Grüzi"       # definition with quotation mark
greeting3 = 'Ciao'
greeting4 = 'Allegra'
```

- *concatenate* strings with '+':

```
>>> name = 'Seppl'
>>> print(greeting1 + name)
BonjourSeppl

#Better, introduce a space:
>> print(greeting1 +" "+ name)          # explicitly control spaces
Bonjour Seppl
>> print(greeting2,name)                # print always introduces a space per ','
Grüzi Seppl
```

- *repetition* with '*'

```
>>> print(3*greeting3)
CiaoCiaoCiao
```

Briefly back to numbers

```
>>> 2**99          # 2 to the power of 99
633825300114114700748351602688

>>> len(str(2**100000))  # how many digits in a BIG number?
30103
```

Note: nested evaluation from left to right (with temporary result along the way)!

Quiz

What is the output of?

```
>>> -5//2
```

Quiz

What is the output of?

```
>>> -5//2
-3                  # i.e., greatest integer smaller than -2.5
```

Lists

- ordered sequences of objects
- accessible by index
- have no *fixed size* and are very flexible
- a list is denoted by *square brackets* []

Three examples:

```
[0,1,2,3]
[0,'abc']                                  # lists can have mixed types; here int + str
['EPFL','is','in',['Lausanne','VD']]       # they support arbitrary nesting
```

Sequence operations (on lists)

NOTE: Lists are sequence objects (just like strings and tuples) and thus behave the same.

```
>>> empty_list=[]     #empty list
>>> L = ['EPFL','is',1,['Lausanne','Geneva']]
>>> len(L)      # length of L. Here 4, not 5!
4

>>> L[0]
'EPFL'
>>> L[0]*3
'EPFLEPFLEPFL'
>>> L[2]*3
3                           # polymorphism at play!
>>> L[3]
['Lausanne','Geneva']       # evaluates to another list,
>>> L[-1]                   # evaluates the last element; L[-2] to second to last, etc.
['Lausanne','Geneva']
>>> L[4]                    # L does not have as many objects!
... error ...
IndexError: list index out of range
```

Lists are mutable

- assigning an element, changes the value!

```
>>> L = ['EPFL','is',1,['Lausanne','Geneva']]
>>> L[0]='MIT'
>>> print(L)
['MIT','is',1,['Lausanne','Geneva']]
```

Type-specific operations on lists

- adding and removing elements from a list (lists are mutable!)

```
>>> L = [0,1,2,3]
>>> L.append(4)            # now the list is [0,1,2,3,4]
>>> print(L)
[0, 1, 2, 3, 4]
>>> L.pop(1)               # remove the 1st element and return value
1
>>> print(L)
[0, 2, 3, 4]
```

Notes:

- like everything in Python, lists are objects. Objects have data!
- and objects have methods and functions, e.g. `append`, `pop`, ...

How to learn about type-specific operations?

- Check out the standard library reference
- stackoverflow ;)
- ask in the Python shell

```
>>> L = [0,1,2,3]
>>> dir(L)                # ommitting some of the options below!
['__add__',...., 'append','clear','copy','count','extend','index','insert','pop',
 'remove','reverse','sort']
>>> help(L.pop)           # alternatively in ipython you can call: L.pop?
Help on built-in function pop:

pop(index=-1, /) method of builtins.list instance
    Remove and return item at index (default last).

    Raises IndexError if list is empty or index is out of range.
```

Dictionaries

- collection of key-value pairs that maps from keys to values.
- the keys can be any immutable type, and the values can be any type.
- like lists they can also be mixed and nested
- a dict is denoted by *curly brackets* {}

A first example: a simple translation system

```
>>> english2spanish = {}
>>> type(english2spanish)
<class 'dict'>
>>> english2spanish['cat'] = 'gato/a'    # defining key-value pairs
>>> # Here str -> str (but can be mixed)
>>> english2spanish['dog'] = 'perro/a'
>>> english2spanish['fox'] = 'zorro'

# Using our translation system:
>>> animal = 'fox'
>>> print("What is "+animal+" in spanish?", english2spanish[animal],', tío/a!')
What is fox in spanish? zorro, tío/a!
>>> # great, but bad spelling! This is easy to fix....
>>> print("What is "+animal+" in spanish?", \
 '¡'+english2spanish[animal].capitalize()+', tío/a!')
What is fox in spanish? ¡Zorro, tío/a!
```

Today's summary

- we encountered several of the built-in objects: `numbers`, `strings`, `lists` and `dictionaries`
- we learned about immutability

Try out the commands in the Python shell/notebooks! Practice is key.

You will dig more into these topics in the exercises (hint: we saw if/else)

Learning by trying it out

slidev — IPython: alex/slidev — ipython — 123×29

```
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev %
(base) alex@alexs-MacBook-Air-2 slidev % ipython
Python 3.9.12 (main, Apr  5 2022, 01:52:34)
Type 'copyright', 'credits' or 'license' for more information
IPython 8.15.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: 2*4
Out[1]: 8

In [2]: 3**4
Out[2]: 81

In [3]: 33*"exciting"+"python"
Out[3]: 'excitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingex
citingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcitingexcit
ingexcitingexcitingexcitingpython'

In [4]:
```

Open your own session with Ipython

Questions?

After lunch:

- Monday 13 - 15: exercises (5 groups)
  - CO4 ← A-B + Y + Z
  - CO5 ← C-F+V
  - CO260 ← G-L+W
  - CO6 ← M-P
  - CO023 ← Q - U
- Monday 15:15 - 16: my office hours at SV 2811 (not today, but email me: alexander.mathis@epfl.ch)