Programming for AMEP

Mathematica & Python

Lecturer: Robert Spreeuw,

TAs: Yu Chih Tseng, Maxim Zewe

Lecturer: Clelia de Mulatier,

TAs: Digvijay, Barsha Bhattacharjee

Tuesday Sept. 12

Programming for AMEP

General Organisation

Schedule:

Tuesday	Wednesday	Friday
13h-15h	15h-17h	11h-13h
Python	Mathematica	Practicals

Fridays: Practicals — discuss organization

Group A = Python
Group B = Mathematica

Two parts: **Week 1 - 5:** Lecture, in-class practicals and assignments (due weeks 2 - 6).

Week 6 - 8: Small programming project — more info closer to the project

Assignments: **deadlines** before the next lecture: i.e., Tuesday at 13h for Python and Wednesday at 15h for Mathematica

must be **uploaded** before the deadline on **canvas**

Grade: **Pass or Fail:** to pass you must validate the exercises of all the modules (more info in the corresponding course)

Advice: During the practical sessions, you will be working on the assignments.
Important: You can directly "pass" your exercises during the practical!

Any questions?

Programming for AMEP

Python Part

Programming for AMEP

Python Part

Goal. Be sufficiently familiar with python and basic programming concepts in computational physics and data analysis, to become independent in your learning of Python.

Plan.

Py 1: Getting familiar with Python

Py 2: Bases of programming in Python

Py 3: Numpy and Linear Algebra

Py 4: Numerical resolution of differential equations

Py 5: Introduction to stochastic simulation

Week 6-8: Small computational project —> list of project given around week 4

[Data visualization with Matplotlib and Handling data with Panda —> resources]

Course Material.

In **Module** section in Canvas

Books.

A Whirlwind Tour of Python, by Jake VanderPlas (O'Reilly). available online (for Py 1 and 2)

Python Data Science Handbook, by Jake VanderPlas (O'Reilly). available online (for Py 3 and 4)

Questions. You can ask you questions:

— during the lecture and lab;

— write to us on Canvas

Assignments.

Due each week just before the next lecture.

Exception for **Py1-Lab.py**

Getting Familiar with Python

Tuesday Sept. 12

Today.

1: Working with Python
2: Python Synthax
3: Variables and Objects
4: Types and Operations
5: Conditional Statements

Course Material.

In **Module 1** section in Canvas

Books. *A Whirlwind Tour of Python*, by Jake VanderPlas (O'Reilly). available **online**

Questions. You can ask you questions:

— during the lecture and lab;
— write to us on Canvas

Assignments.

Due on Monday, just before the next lecture.

1-
Working with Python

Conception.

Python was conceived in the late 1980s by Guido van Rossum at Centrum Wiskunde & Informatica (CWI)

Setting up Python

Python3. Please let us know if you haven't managed to install Python3 on your laptop

```
$ python3 -- version          —> Check your version, and location
```

Tips for MacOs Monterey

pip. Python's default package manager. Allows you to install packages (numpy, pandas, tensorflow, etc.) and their dependencies.

```
$ pip -- version              —> Check your version, and location
$ pip3 -- version             —> or that…
$ python3 -m pip -- version
```

Please let us know if you issue finding the "correct" call command

Other package managers. Ex. Anaconda, Pyenv —-> more info

Working with Python

Python Interpreter. The most basic way to execute Python code is line by line. Used this way, python can be handy when you need a quick access to a calculator. —> **Try it!**

For more advanced calculation? sqrt(2) ???

Working with Python

Python Interpreter. The most basic way to execute Python code is line by line. Used this way, python can be handy when you need a quick access to a calculator. —> **Try it!**

Import the python module math for more advanced calculation:

```
>>> import math
>>> math.sqrt(2.)
```

To quit the interpreter. `>>> quit()` Or: CTRL+D

Working with Python

Python Interpreter. The most basic way to execute Python code is line by line. Used this way, python can be handy when you need a quick access to a calculator. —> **Try it!**

Import the python module math for more advanced calculation:

```
>>> import math
>>> math.sqrt(2.)
```

Python scripts. for more complicated programs it is more convenient to save code in a file, and execute it all at once. Create this script and run it:

# file: test_script.py

```
print("Running ‘test_script.py’:”)
x = 5
print("x = ", x)
print("3*x = ", 3 * x)
```

! You must first find the location of your script from your terminal !

Modify your script and run it again.

Working with Python

Python Interpreter. The most basic way to execute Python code is line by line. Used this way, python can be handy when you need a quick access to a calculator. —> **Try it!**

Import the python module math for more advanced calculation:

```
>>> import math
>>> math.sqrt(2.)
```

Python scripts. for more complicated programs it is more convenient to save code in a file, and execute it all at once. Create this script and run it:

```
# file:  test_script.py
print("Running 'test_script.py':")
x = 5
print("x = ", x)
print("3*x = ", 3 * x)
```

Jupyter Notebook. (Link) Useful hybrid of the interactive terminal and the self-contained script.

```
$ pip install notebook       —> to install the package.  Or $ python3 -m pip [...]
$ jupyter notebook           —> to start Jupiter Notebook
CTRL + C                     —> to quit
```

everything **works locally** (no need for internet)

>> Open Py1_Lab-.ipynp   Go through the **Jupyter notebook tips (5min)**

Organisation of the notebooks

Part 1 = lecture

Part 2 = Homework assignment

2 - Python Synthax

Python Language Syntax

What do you notice about the syntax?

```
# define two variables a and b
a=10;    b=20

# define c and x
c = 3*(b-a)
d = 1 + 2 + 3 + 4 +\
    5 + 6 + 7 + 8

# silly test:
if c < 0:
    print(b)
elif a < 0 and b < 0:
    print(c)
else:
    print(d)

# perfect end:
s1 = 'this is a string'
s2 = "another string with double quotes"
s3 = ''' string can span
         multiple lines '''
```

Python Language Syntax

End a statement:

a **semicolon** ends a statement,
But is not necessary; **newline** also ends a statement

Identifiers:

names of variables, functions, modules,
start with a letter or "_" and are **case-sensitive**

Indentation:

used **to organize** the code

Comments starts with #

Continuation of a statement to a new line with \

Strings can be denoted with ', ", ''', """

```
# define two variables a and b
a=10;     b=20

# define c and x
c = 3*(b-a)
d = 1 + 2 + 3 + 4 +\
    5 + 6 + 7 + 8

# silly test:
if c < 0:
    print(b)
elif a < 0 and b < 0:
    print(c)
else:
    print(d)

# perfect end:
s1 = 'this is a string'
s2 = "another string with double quotes"
s3 = ''' string can span
         multiple lines '''
```

3 -
Variables and Objects

Variables, Objects, Types, Operations

Python variables and objects

Variable. name given to a memory location, where will be stored a value. A variable is created the moment we assign a first value to it.

```
>>> del(x)
>>> print(x)
```

Python variables and objects

Variable. name given to a memory location, where will be stored a value. A variable is created the moment we assign a first value to it.

```
>>> x = 10
>>> print(x)
10
```

```
>>> del(x)
>>> print(x)
NameError: name 'x' is not defined
```

Python variables and objects

Variable. name given to a memory location, where will be stored a value. A variable is created the moment we assign a first value to it.

```
>>> x = 10
>>> x = 'hello'
>>> print(x)
?
```

```
>>> del(x)
>>> print(x)
NameError: name 'x' is not defined
```

Python variables and objects

Variable. name given to a memory location, where will be stored a value. A variable is created the moment we assign a first value to it.

```
>>> del(x)
>>> print(x)
NameError: name 'x' is not defined
```

```
>>> x = 10
>>> x = 'hello'
>>> print(x)
hello
```

You can change the value and type of a variable without any issue.

Variables are pointers to objects.   While objects have a specific type, variables can point to objects of any type.

>>> Q1

Python variables and objects

Variable. name given to a memory location, where will be stored a value. A variable is created the moment we assign a first value to it.

```
>>> del(x)
>>> print(x)
NameError: name 'x' is not defined
```

```
>>> x = 10
>>> x = 'hello'
>>> print(x)
hello
```

You can change the value and type of a variable without any issue.

Variables are pointers to objects. While objects have a specific type, variables can point to objects of any type.

>>> Q1

```
>>> a is b
```

"**is**" test if two objects are the same

Python variables and objects

Variable. name given to a memory location, where will be stored a value. A variable is created the moment we assign a first value to it.

```
>>> del(x)
>>> print(x)
NameError: name 'x' is not defined
```

```
>>> x = 10
>>> x = 'hello'
>>> print(x)
hello
```

You can change the value and type of a variable without any issue.

Variables are pointers to objects. While objects have a specific type, variables can point to objects of any type. **>>> Q1**

Object. Each object has a **type**, and can have **attributes** and **methods**.

Attributes: contain data.

```
>>> x = 4.5
>>> print(x.real, "+", x.imag, 'i')
4.5 + 0.0 i
```

Methods: functions that can act on the attributes of the object

```
>>> x = 4.5
>>> x.is_integer()
False
>>> x = 4.0
>>> x.is_integer()
True
```

Accessed via the dot syntax:

>>> Q2

4 -
Types and Operations

Types

Simple build-in types: number types (integer, float and complex), Boolean, and string.

Type	Example	Description
int	`x = 1`	Integers (i.e., whole numbers)
float	`x = 1.0`	Floating-point numbers (i.e., real numbers)
complex	`x = 1 + 2j`	Complex numbers (i.e., numbers with real and imaginary part)
bool	`x = True`	Boolean: True/False values
str	`x = 'abc'`	String: characters or text
NoneType	`x = None`	Special object indicating nulls

More complex types: built-in data structure. —> next lecture.

Variable types.

```
>>> a=1
>>> type(a)
```

```
>>> a="hello"
>>> type(a)
```

Types

Simple build-in types: number types (integer, float and complex), Boolean, and string.

Type	Example	Description
int	`x = 1`	Integers (i.e., whole numbers)
float	`x = 1.0`	Floating-point numbers (i.e., real numbers)
complex	`x = 1 + 2j`	Complex numbers (i.e., numbers with real and imaginary part)
bool	`x = True`	Boolean: True/False values
str	`x = 'abc'`	String: characters or text
NoneType	`x = None`	Special object indicating nulls

More complex types: built-in data structure. —> next lecture.

Variable types. Is not fixed!

```
>>> a=1
>>> type(a)
int
```

```
>>> a="hello"
>>> type(a)
str
```

Types

Simple build-in types: number types (integer, float and complex), Boolean, and string.

Type	Example	Description
int	`x = 1`	Integers (i.e., whole numbers)
float	`x = 1.0`	Floating-point numbers (i.e., real numbers)
complex	`x = 1 + 2j`	Complex numbers (i.e., numbers with real and imaginary part)
bool	`x = True`	Boolean: True/False values
str	`x = 'abc'`	String: characters or text
NoneType	`x = None`	Special object indicating nulls

More complex types: built-in data structure. —> next lecture.

Variable types. Is not fixed!

```
>>> a=1
>>> type(a)
int
```

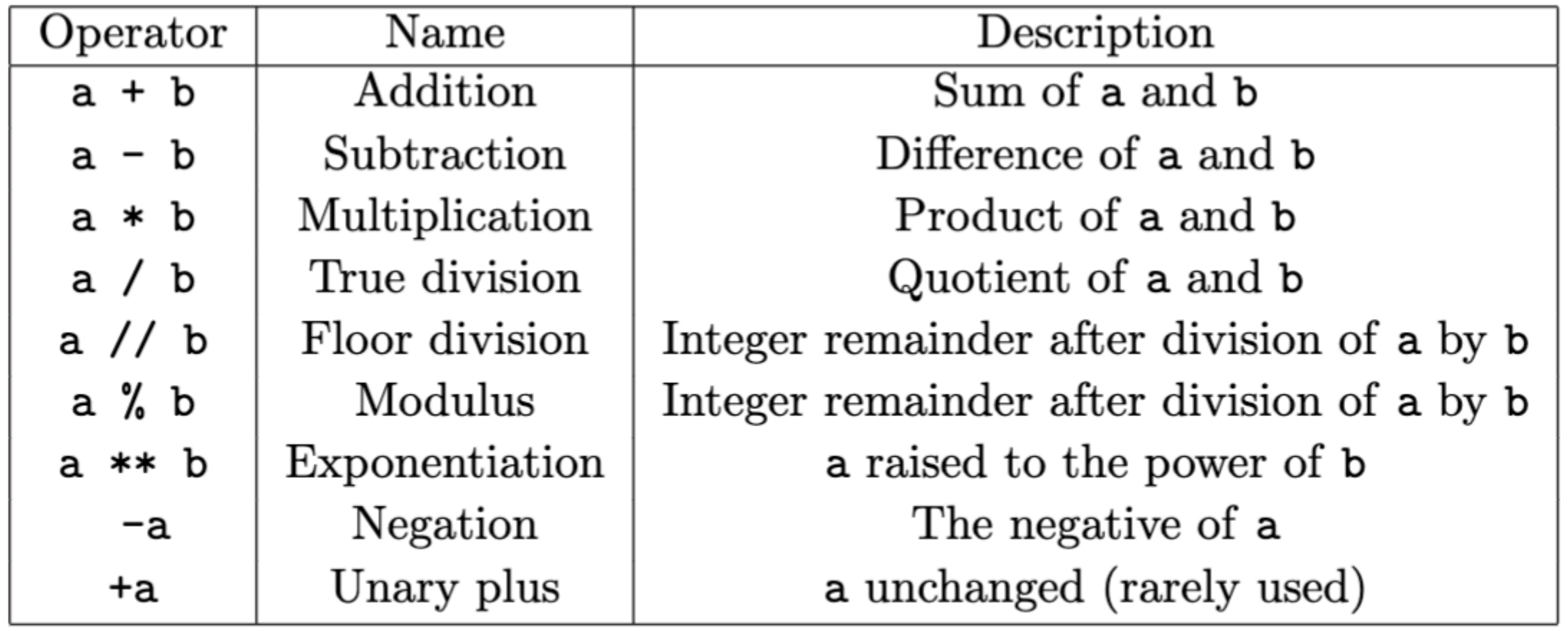
```
>>> a="hello"
>>> type(a)
str
```

Integers	Float

python2: encoded on 32 or 64 bits
python3: no limit!!!!! More info about Bignum here
However lose computational efficiency
`>>> int(3.6)` ?
`>>> round(3.6)` ?
	Most of the time on 64 bits. More info here
	Largest value: max=1.7976931348623157e+308
	Smallest value: min=2.2250738585072014e-308
	`>>> import sys`
	`>>> sys.float_info.max`
	`>>> sys.float_info.min`

Operations

Numerical operators:



Operator	Name	Description
`a + b`	Addition	Sum of `a` and `b`
`a - b`	Subtraction	Difference of `a` and `b`
`a * b`	Multiplication	Product of `a` and `b`
`a / b`	True division	Quotient of `a` and `b`
`a // b`	Floor division	Integer remainder after division of `a` by `b`
`a % b`	Modulus	Integer remainder after division of `a` by `b`
`a ** b`	Exponentiation	`a` raised to the power of `b`
`-a`	Negation	The negative of `a`
`+a`	Unary plus	`a` unchanged (rarely used)

Other operations:

Bitwise operations, **assignment** operations, **comparison** operations, **Boolean** operations, and **Identity** and **membership** operations.

See here

Operations

Numerical operators:

Operator	Name	Description
`a + b`	Addition	Sum of `a` and `b`
`a - b`	Subtraction	Difference of `a` and `b`
`a * b`	Multiplication	Product of `a` and `b`
`a / b`	True division	Quotient of `a` and `b`
`a // b`	Floor division	Integer remainder after division of `a` by `b`
`a % b`	Modulus	Integer remainder after division of `a` by `b`
`a ** b`	Exponentiation	`a` raised to the power of `b`
`-a`	Negation	The negative of `a`
`+a`	Unary plus	`a` unchanged (rarely used)

Other operations:

Bitwise operations, **assignment** operations, **comparison** operations, **Boolean** operations, and **Identity** and **membership** operations.

See here

Python2 VS Python3:

In python2: >>> 3 / 2 ?

In python3: >>> 3 / 2 ?

Operations

Numerical operators:

Operator	Name	Description
`a + b`	Addition	Sum of `a` and `b`
`a - b`	Subtraction	Difference of `a` and `b`
`a * b`	Multiplication	Product of `a` and `b`
`a / b`	True division	Quotient of `a` and `b`
`a // b`	Floor division	Integer remainder after division of `a` by `b`
`a % b`	Modulus	Integer remainder after division of `a` by `b`
`a ** b`	Exponentiation	`a` raised to the power of `b`
`-a`	Negation	The negative of `a`
`+a`	Unary plus	`a` unchanged (rarely used)

Other operations:

Bitwise operations, **assignment** operations, **comparison** operations, **Boolean** operations, and **Identity** and **membership** operations.

See here

Python2 VS Python3:

In python2: >>> 3 / 2 ? 1

Same in C, fortran, …

In python3: >>> 3 / 2 ? 1.5

>>> Q6 and Q7

Operations

Numerical operators:

Operator	Name	Description
`a + b`	Addition	Sum of `a` and `b`
`a - b`	Subtraction	Difference of `a` and `b`
`a * b`	Multiplication	Product of `a` and `b`
`a / b`	True division	Quotient of `a` and `b`
`a // b`	Floor division	Integer remainder after division of `a` by `b`
`a % b`	Modulus	Integer remainder after division of `a` by `b`
`a ** b`	Exponentiation	`a` raised to the power of `b`
`-a`	Negation	The negative of `a`
`+a`	Unary plus	`a` unchanged (rarely used)

Other operations:

Bitwise operations, **assignment** operations, **comparison** operations, **Boolean** operations, and **Identity** and **membership** operations.

See here

Python2 VS Python3:

In python2: `>>> 3 / 2` ? 1

Same in C, fortran, …

In python3: `>>> 3 / 2` ? 1.5

>>> Q6 and Q7

Example:

```
>>> a = 10
>>> b = "Test"
>>> print(a+b)
```

Operations

Numerical operators:

Operator	Name	Description
`a + b`	Addition	Sum of `a` and `b`
`a - b`	Subtraction	Difference of `a` and `b`
`a * b`	Multiplication	Product of `a` and `b`
`a / b`	True division	Quotient of `a` and `b`
`a // b`	Floor division	Integer remainder after division of `a` by `b`
`a % b`	Modulus	Integer remainder after division of `a` by `b`
`a ** b`	Exponentiation	`a` raised to the power of `b`
`-a`	Negation	The negative of `a`
`+a`	Unary plus	`a` unchanged (rarely used)

Other operations:

Bitwise operations, **assignment** operations, **comparison** operations, **Boolean** operations, and **Identity** and **membership** operations.

See here

Python2 VS Python3:

In python2: `>>> 3 / 2` ? 1

Same in C, fortran, …

In python3: `>>> 3 / 2` ? 1.5

>>> Q6 and Q7

TypeError:

```
>>> a = 10
>>> b = "Test"
>>> print(a+b)

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

5 - Comparison operators, Boolean logic, Conditional Statements

Operations

Comparison operators:

Operation	Description	Operation	Description
`a == b`	`a` equal to `b`	`a != b`	`a` not equal to `b`
`a < b`	`a` less than `b`	`a > b`	`a` greater than `b`
`a <= b`	`a` less than or equal to `b`	`a >= b`	`a` greater than or equal to `b`

Boolean operators: and or not

>>> Q9

Operations

Comparison operators:

Operation	Description	Operation	Description
`a == b`	`a` equal to `b`	`a != b`	`a` not equal to `b`
`a < b`	`a` less than `b`	`a > b`	`a` greater than `b`
`a <= b`	`a` less than or equal to `b`	`a >= b`	`a` greater than or equal to `b`

Boolean operators: and or not

>>> Q9

a is equal to 10:	`a==10`
a is larger than 15 and smaller than b:	`a > 15 and a < b`
a is larger than 15 or smaller than b:	`a > 15 or a < b`
a is not equal to b:	`a != b`
a is not larger than b:	`not a > b`
a is odd:	`a%2 == 1`

>>> Q10, Q11

Operations

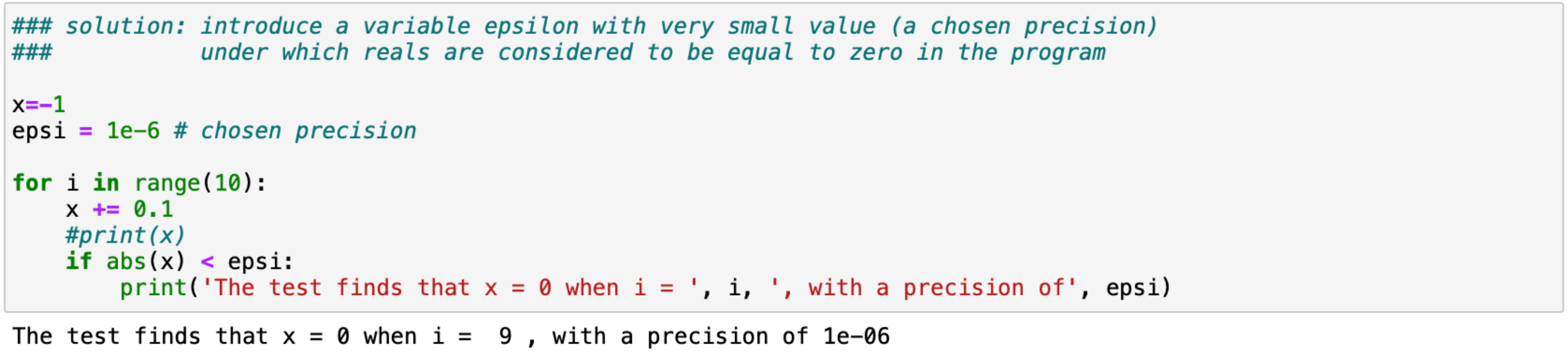
>>> Q10

```
### solution: introduce a variable epsilon with very small value (a chosen precision)
###           under which reals are considered to be equal to zero in the program

x=-1
epsi = 1e-6 # chosen precision

for i in range(10):
    x += 0.1
    print(x)
if x < epsi:
    print('The test finds that x = 0 to a precision of', epsi)
else:
    print('The test finds that x is different from 0')
```

```
-0.9
-0.8
-0.7000000000000001
-0.6000000000000001
-0.5000000000000001
-0.40000000000000013
-0.30000000000000016
-0.20000000000000015
-0.10000000000000014
-1.3877787807814457e-16
The test finds that x = 0 to a precision of 1e-06
```

Operations

>>> Q11



```python
### solution: introduce a variable epsilon with very small value (a chosen precision)
###           under which reals are considered to be equal to zero in the program

x=-1
epsi = 1e-6 # chosen precision

for i in range(10):
    x += 0.1
    #print(x)
    if abs(x) < epsi:
        print('The test finds that x = 0 when i = ', i, ', with a precision of', epsi)
```

```
The test finds that x = 0 when i =  9 , with a precision of 1e-06
```

Conditional Statements

Contraction of "else if"

```
x = -15

if x == 0:
    print(x, "is zero")
elif x > 0:
    print(x, "is positive")
elif x < 0:
    print(x, "is negative")
else:
    print(x, "is unlike anything I've ever seen...")
```

Conditional Statements

```
x = -15

if x == 0:
    print(x, "is zero")
elif x > 0:
    print(x, "is positive")
elif x < 0:
    print(x, "is negative")
else:
    print(x, "is unlike anything I've ever seen...")
```

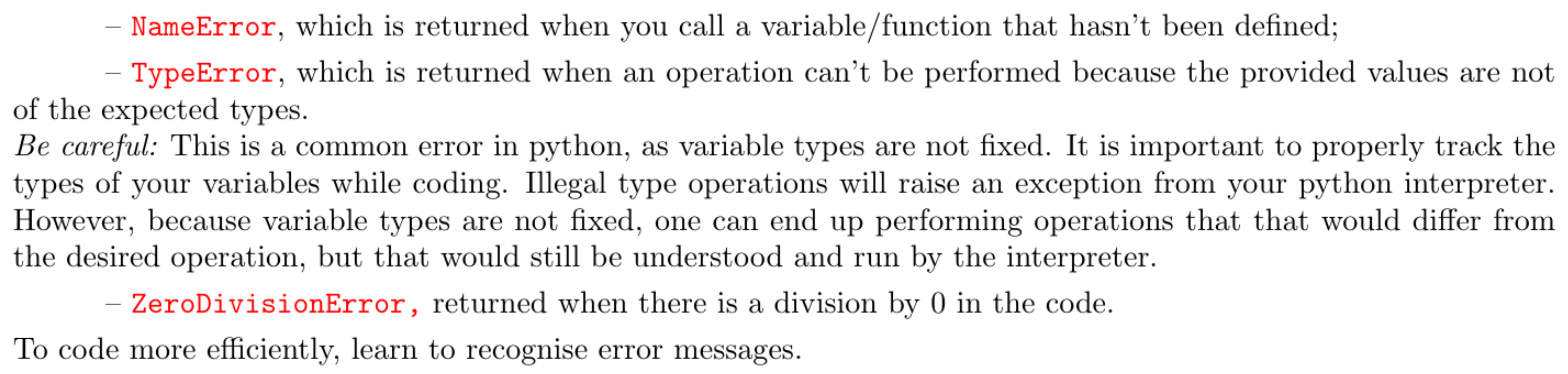
Contraction of "else if"

```
-15 is negative
```

Others

Illegal operations

An operation that is not authorized or not understood by the python interpreter.
The operation is terminated and the interpreter returns an error message.



– `NameError`, which is returned when you call a variable/function that hasn't been defined;

– `TypeError`, which is returned when an operation can't be performed because the provided values are not of the expected types.
Be careful: This is a common error in python, as variable types are not fixed. It is important to properly track the types of your variables while coding. Illegal type operations will raise an exception from your python interpreter. However, because variable types are not fixed, one can end up performing operations that that would differ from the desired operation, but that would still be understood and run by the interpreter.

– `ZeroDivisionError,` returned when there is a division by 0 in the code.

To code more efficiently, learn to recognise error messages.

Python modules

Python standard library contains useful tools for a wide range of tasks.

```
>>> import math
>>> math.sqrt(2)
```

On top of this, there is a broad ecosystem of third-party tools and packages that offer more specialized functionality. Here we'll take a look at importing standard library modules, tools for installing **third-party modules**, and a description of how you can make your own modules.

```
>>> import numpy as np
>>> np.sqrt(2)
```

Python2 VS Python3

Good programming habits?

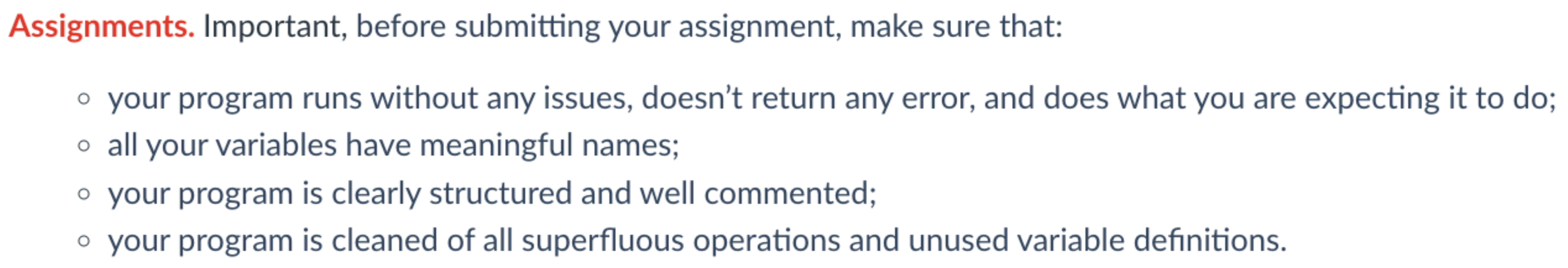
Good programming habits?

- **Give meaningful names** to your variables and functions.
- **Add comments** across your code. To comment a part of a line, precede it with the character `#`. For example:
  `>>> a=1 # a is an integer`
- Decompose your programs in small functions, rather than working out a large program; i.e., defined intermediate functions, as well as variables.
- **Clarity:** Clean up useless variables and operations. In some situations, you may wish to use a very long command that doesn't fit on one line. For such cases, you can end a line with a backslash. Python will then continue reading the next line as part of the same command. Try this:
- **Test!** Don't wait for having written the whole program to start testing; test each function as soon as they are written, and test sub-part of a function if the function is long;
- Errors: read and try to understand the error messages; google error messages that you don't understand;
- Bugs: comment parts of the program and run test on smaller pieces of the program to figure out where a bug or an error comes from. Use debugging function of Jupyter notebook.
- **look for information/help in the documentation and online.** A python documentation is available online at https://www.python.org/doc/. You will also often find answers to your questions on the web or by visiting stackoverflow.com. You can also get help directly from Python for build-in functions by using the `help()` command. For instance to ask information about the function `round`, type `help(round)` at the command prompt. To quite the "help", press "q";
- More advanced: Remember to free out memory.

The Zen of Python, by Tim Peters. Type ***import this*** in your python interpreter.

Assignment for Friday

Submit assignment on Canvas. **Py1-Lab.py** is Due on Friday evening.



Assignments. Important, before submitting your assignment, make sure that:

- your program runs without any issues, doesn't return any error, and does what you are expecting it to do;
- all your variables have meaningful names;
- your program is clearly structured and well commented;
- your program is cleaned of all superfluous operations and unused variable definitions.

Questions. You can ask you questions:

— during the lecture and lab;

— write to us on Canvas