جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL

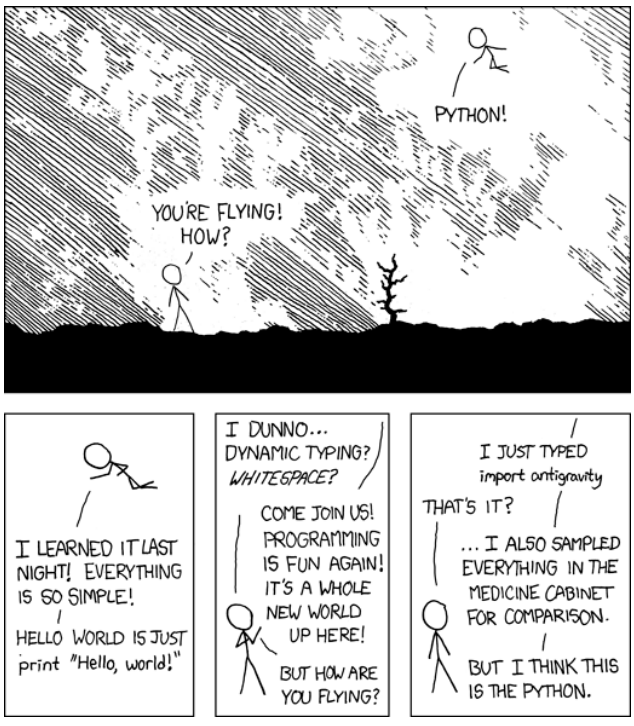
المدرسة الوطنية العليا للفنون و المهن
ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS

Python Programming
- Introduction -

EL Moukhtar ZEMMOURI

ENSAM – Meknès

Version 1.0 – 2019



https://xkcd.com/353/

What is Python?

- Very high level programming language
- Object oriented
- High level and powerful built in data structures
- Dynamic typing and dynamic binding
- Support modules and packages
- Open source and free ☺
- Dynamic community and rapid development ☺
- Simple and easy to learn ☺

What is Python? Executive Summary

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code re-use. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

https://www.python.org/doc/essays/blurb/

A Brief Timeline of Python

- 1989 : Implementation started by Guido Von Rossum
- 1991 : first version 0.9.0 released
- 1994 : version 1.0.0
- 2001 : Python Software Foundation (https://www.python.org/psf/)
- 2008 : version 2.6 and 3.0
- ... 3.x and 2.x coexist, but incompatible
  - 2.7 is end-of-life release

Running Python

- Python has two basic modes: **interactive** and **script**.
- Interactive using command line shell
  - Mainly during development

```
In [1]: print("Hello World !")
Hello World !

In [2]: a = 10

In [3]: b = 20

In [4]: a+b
Out[4]: 30

In [5]:
```

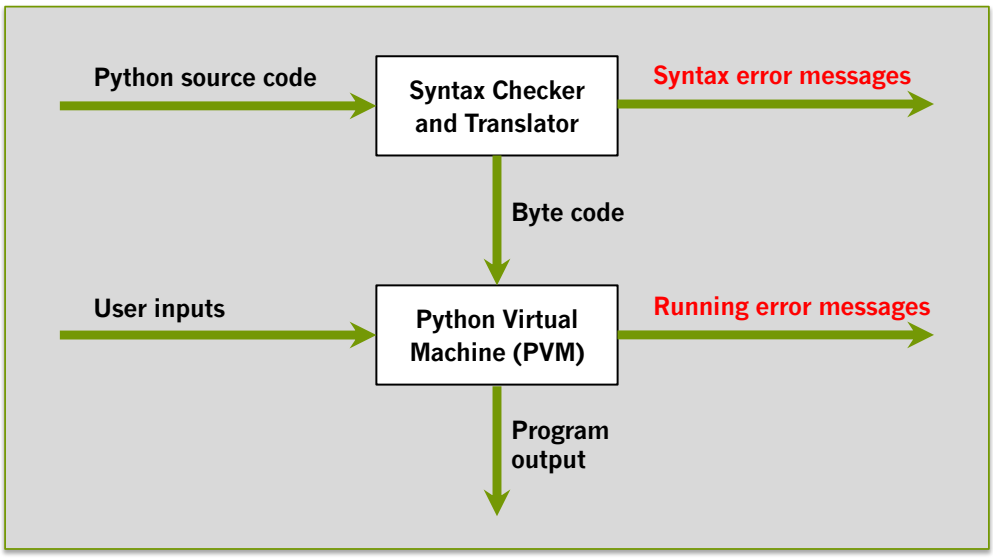
- Script : write code in a .py file and run
  - Mainly for long runs

example.py

```
1  print("Hello World !")
2  n = int(input("Give an integer : "))
3  if n%2 == 0:
4      print(n, 'is an even number')
5  else :
6      print(n, 'is an odd number')
7
```

```
In [6]: run example.py
Hello World !
Give an integer : 12
12 is an even number
```

Python interpretation process



Basic Datatypes and operators

Basic Datatypes

- numbers.Number
  - Integral
    - int
    - bool
  - Real
    - float
  - Complex
    - complex

Integers

- **int :** represent numbers in an *unlimited range*, subject to available (virtual) memory only.
  - Example : run >>> 123456789 ** 100
- Literals representation:
  - Decimal : default representation 1234
  - Binary : begin with **0b**
  - Octal : begin with **0o**
  - Hexadecimal : begin with **0x**

```
In [1]: 1234
Out[1]: 1234

In [2]: 0b10101
Out[2]: 21

In [3]: 0o1234
Out[3]: 668

In [4]: 0x1234
Out[4]: 4660

In [5]:
```

Integers

- Operators :
  - +, –, *, / (real division), %, // (integer division), ** (power)
  - +=, –=, *=, /=, //=, %=, … (compound-assignment)
    - Var op= expression ➔ Var = Var op expression
    - Examples : **`S += i; i += 1; i-=1`** **`(note there is no ++ in python)`**
  - Bitwise operations
    - & (and), | (or), ^ (exclusive or), ~ (ones' complement)
    - >> (right shift) , << (left shift)
  - Note:
    - **`divmod`** returns a tuple (q, r) of the Euclidian division.
    - Example : **`q, r = divmod (14, 3)`** ➔ **`(q, r) = (4, 2)`**

Integers

- Some built-in functions
  - **`int (x, base=10)`** : Return an integer constructed from a number or string x, or return 0 if no arguments are given
    - **`n = int ('1234')`** ➔ **`n=1234`**
    - **`n = int ('ab')`** ➔ **`ValueError`**
    - **`n = int ('ab', 16)`** ➔ **`n=171`**
  - **`hex(x)`** : Convert an integer number to a lowercase hexadecimal string prefixed with "0x".
    - **`hex (1234)`** ➔ **`'0x4d2'`**
  - **`bin(x), oct(x)`** : Conversion to base 2 and 8 (the results is a string)
  - **`abs(x)`** : Return the absolute value of x.

Real

- **float :** represent double precision floating point numbers.
  - Python does not support single-precision floating point numbers ☺
  - Use of IEEE-754 double precision floating point arithmetic.
  - But depend on machine architecture (and C or Java implementation) for the accepted range and handling of overflow.
- Literals representation:
  - Decimal notation
    - `1.5  0.00015  154321.0`
  - Scientific notation (exponential)
    - `1.5e0  1.5e-4  1.54321e5`

Real

- Operators :
  - +, –, *, / (real division), ** (power)
    - Note : `0 ** 0 = 1` (this is common for programming)
  - +=, –=, *=, /=, //=, %=, ... (compound-assignment)
- Some built-in functions
  - `float(x)` : Return a floating point number constructed from a number or string x.
  - `power(x, y)` : Return x to the power y. Same as x ** y.
  - `round(x, n)` : Return x rounded to n digits precision after the decimal point. If n is omitted or is None, it returns the nearest integer to x.
  - `abs(x)` : Return the absolute value of x.
- math module :
  - sqrt, sin, cos, tan, asin, acos, atan, log, exp, factorial, floor, ciel, trunc, ..., pi, e, inf, nan, ...

Boolean

- **bool :** represent the truth values `False` and `True`.
  - subtype of integer, and False and True behave like the 0 and 1.
- Operators :
  - `and, or, not`
    - With short-circuit execution
- Comparison :
  - `<, >, <=, >=, ==, !=`
  - `is, is not` (object identity comparison)

Truth Value Testing

- Any object can be tested for truth value (ex. for use in `if` or `while` condition).
- The following values are evaluated False:
  - None
  - False
  - zero of any numeric type : `0, 0.0, 0j, Decimal(0), Fraction(0,1)`
  - empty sequences and collections : `'', (), [], {}, set(), range(0)`
- Otherwise, it is True

Complex

- **complex :** represent complex numbers as a pair of double precision floating point numbers.
  - The real and imaginary parts of z are `z.real` and `z.imag`.
  - **`complex (x, y)`** :
    - Create a complex number with the value **`x + yj`** or convert a string or number to a complex number.
  - Examples :
    - `z = 1 + 2j`
    - `z = complex (1, 2)`

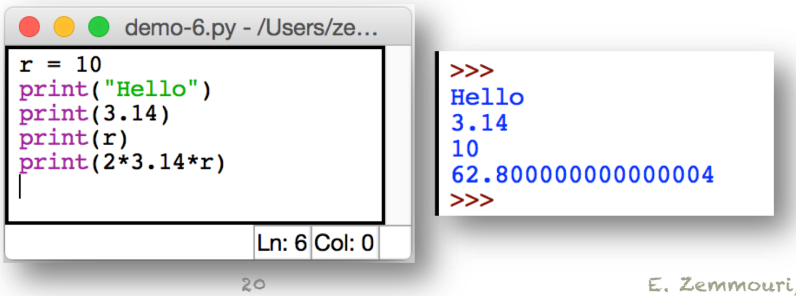
Complex

- Operators:
  - +, –, *, /, **
- Some built-in functions :
  - **`abs(z)`** : Return the the magnitude of z.
  - **`z.conjugate()`** : Return the conjugate of z.
- cmath module : `import cmath`
  - polar(z), rect(*r*, *phi*), exp(z), log(z), sqrt(z), sin, cos, tan, asin, acos, atan ...
  - pi, e, inf, nan, ...

Lecture / écriture

Fonction d'affichage

- La fonction print
  - Permet d'afficher une ou plusieurs valeurs.
    - Une valeur peut être : un message (chaine de caractères constante), une constante, le contenu d'une variable, ou d'une manière générale une expression.
  - print(expression)
    - L'expression sera évaluée, puis le résultat affiché.
  - Exemple :



Fonction d'affichage

- La fonction **print**
  - print(expression, ..., expression, sep=' ', end='\n')
    - sep : chaine de caractères insérée entre deux valeurs successives
      - Par défaut espace.
    - end : chaine de caractères afficher après la dernière valeur.
      - Par défaut retour à la ligne.
  - Exemple : (à exécuter)
    - A = "Hello" ; B = "World"
    - print(A, B)
    - print(A, B, sep = '–')
    - print(A) ; print(B)
    - print(A, end=' ') ; print(B)

Lecture de données

- La fonction **input**
  - Affiche un message invitant l'utilisateur à la saisie
  - Retourne la valeur saisie par l'utilisateur sous forme d'une chaine de caractères
    - En fonction du type souhaité, il faut utiliser un mécanisme de transtypage (exemple : fonction de conversion de type).
  - input("message")

```
>>>
>>> name = input("Enter your name ")
Enter your name Zemmouri
>>> print("Your name is :", name)
Your name is : Zemmouri
>>> n = int(input("Donnez un entier : "))
Donnez un entier : 100
>>> n
100
>>> x = float(input("Donner un réel : "))
Donner un réel : 3.14
>>> x
3.14
>>>
```

Control Flow

Bloc d'instructions

- Les blocs d'instructions sont délimités par l'indentation.
  - En général, le décalage est de 4 espaces.
- Les blocs peuvent être imbriqués:

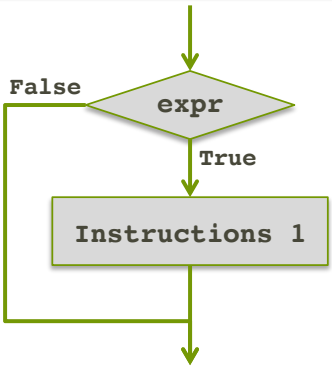
```
Ligne d'entête :
    Instruction 1
    Instruction 2
    ...
    Instruction N
```

```
Ligne d'entête :
    Instruction
    Instruction
    Ligne d'entête :
        Instruction
        Instruction
    Instruction
    Instruction
```

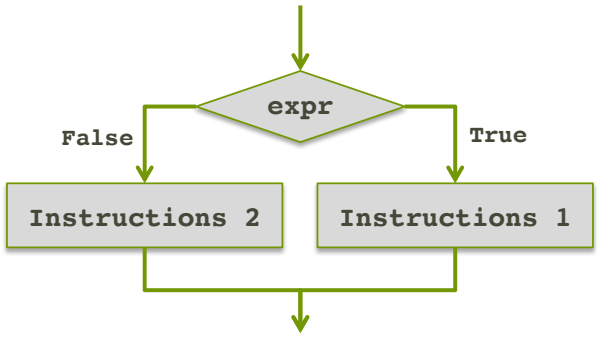
La sélection

- Utilisée pour l'exécution conditionnelle
- Sélection simple :

```
if expression :
    Bloc d'instructions
```

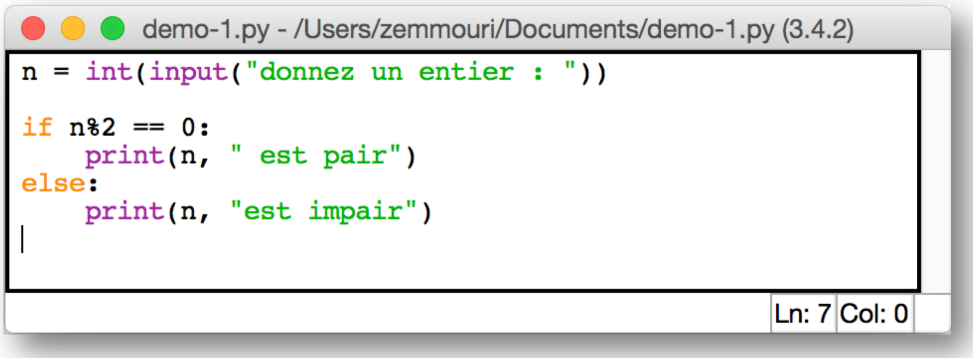



```
if expression :
    Bloc d'instructions 1
else:
    Bloc d'instructions 2
```



La sélection

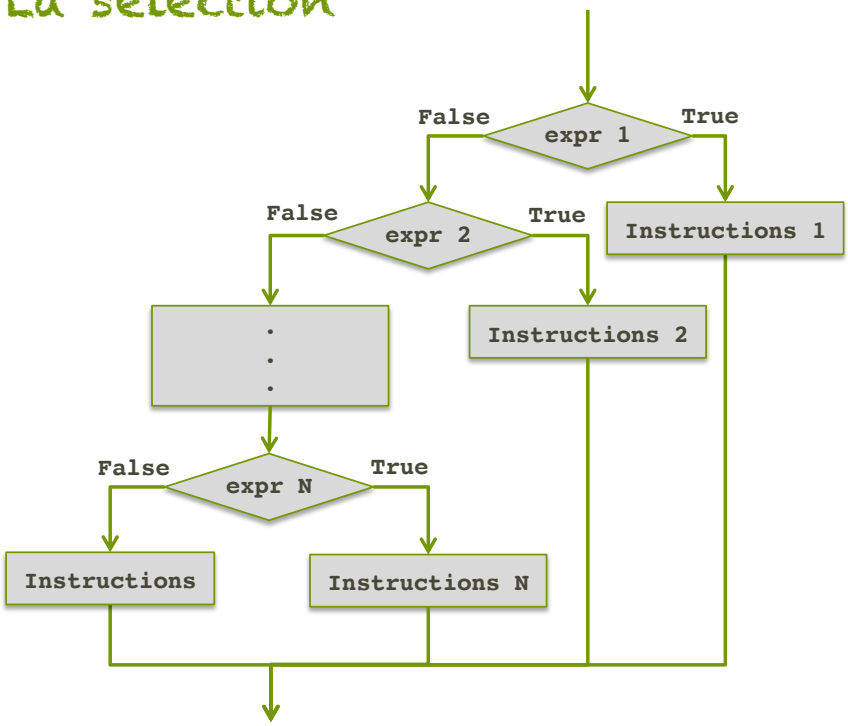
- Sélection simple
  - Exemple : tester si un nombre est pair ou impair



La sélection

- Sélection multiple

```
if expr 1 :
    Bloc d'instructions 1
elif expr 2 :
    Bloc d'instructions 2
elif expr 3 :
    Bloc d'instructions 3
.
.
.
else:
    Bloc d'instructions
```



La sélection

- Sélection multiple
  - Exemple : afficher le grade (mention) correspondant à une moyenne

demo-2.py - /Users/zemmouri/Documents/demo-2.py (3.4.2)

```
moy = float(input("donnez la moyenne entre 0 et 20 : "))

if moy >= 18 :
    grade = "Excellent"
elif moy >= 16:
    grade = "Très bien"
elif moy >= 14:
    grade = "Bien"
elif moy >= 12:
    grade = "Assez bien"
elif moy >= 10:
    grade = "Passable"
else:
    grade = "Echec"

print("Le grade obtenu est : ", grade)
```

Ln: 16 Col: 38

La répétition

- La boucle **for**
  - Permet de répéter un bloc d'instructions un nombre prédéfini de fois

```
for var in range(start, stop, step):
    Instruction 1
    ...
    Instruction N
```

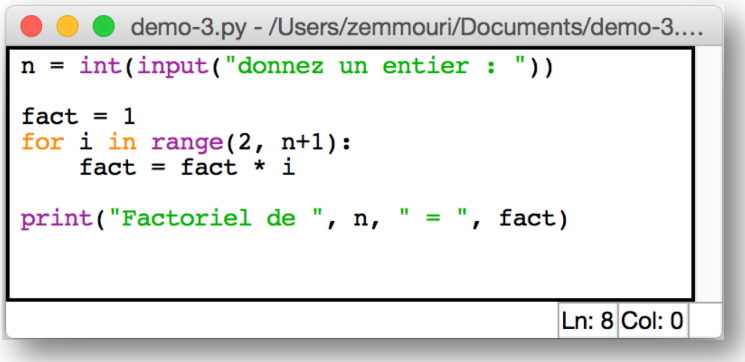
- **range** (start, stop, step) :
  - Retourne une séquence de nombre de start à stop (exclu) avec un pas step
  - Par défaut start = 0 et step = 1

```
>>> for i in range(1, 5):
        print(i)


1
2
3
4
>>>
```


La répétition

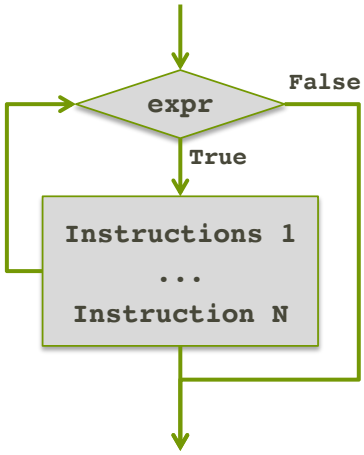
- La boucle **for**
  - Exemple : calcul du factoriel d'un entier



La répétition

- La boucle **while**
  - Permet de répéter un bloc d'instructions tant que une condition est vraie

```
while expression:
    Instruction 1
    ...
    Instruction N
```



La répétition

- La boucle **while**
  - Exemple : calcul du pgcd de deux entiers

demo-4.py - /Users/zemmouri/Documents/demo-4.py (3.4.2)

```
a = int(input("donnez le premier nombre : "))
b = int(input("donnez le deuxième nombre : "))

r = a%b
while r != 0:
    a = b
    b = r
    r = a%b

print("le pgcd est ", b)
```

Ln: 11 Col: 0

La répétition

- La boucle **while**
  - Exemple : somme de nombres saisis au clavier

demo-5.py - /Users/zemmouri/Documents/demo-5.py (3.4.2)

```
sum = 0.0
data = input("Enter a number or just enter to quit: ")
while data != "":
    number = float(data)
    sum += number
    data = input("Enter a number or just enter to quit: ")
print("The sum is", sum)
```

Ln: 8 Col: 0

break et continue

- Utilisées dans une boucle **for** ou **while**.
  - **break** : permet de sortir de la boucle
  - **continue** : permet de passer à l'itération suivante de la boucle

```
>>>
>>> for n in range(2, 10):
        prime = True
        for i in range(2, n):
            if n%i == 0:
                prime = False
                break
        if prime :
            print(n, "is a prime number")


2 is a prime number
3 is a prime number
5 is a prime number
7 is a prime number
>>>
```

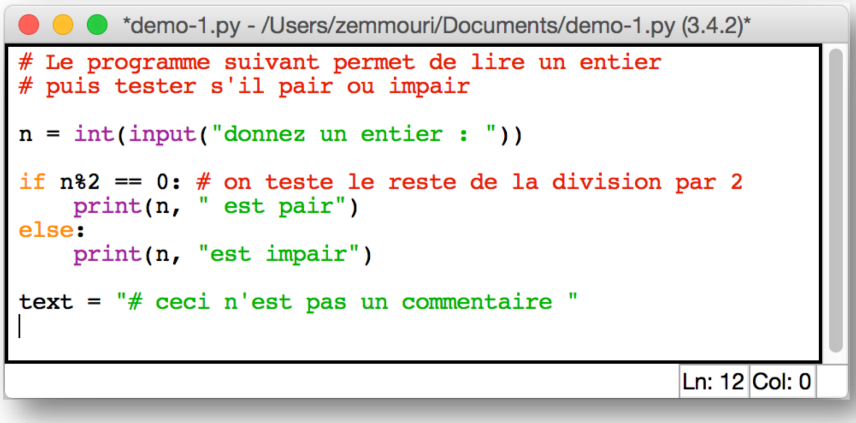
```
>>>
>>> for c in "bla bla bla":
        if c == "l":
            continue
        print(c, end='')


ba ba ba
>>>
```

Commentaires

Les commentaires

- Les commentaires permettent de documenter un programme.
- Un commentaire en Python commence par **#**



Multiline comments

- Put text between " " " and " " ”
  - Or between ' ' ' and ' ' '