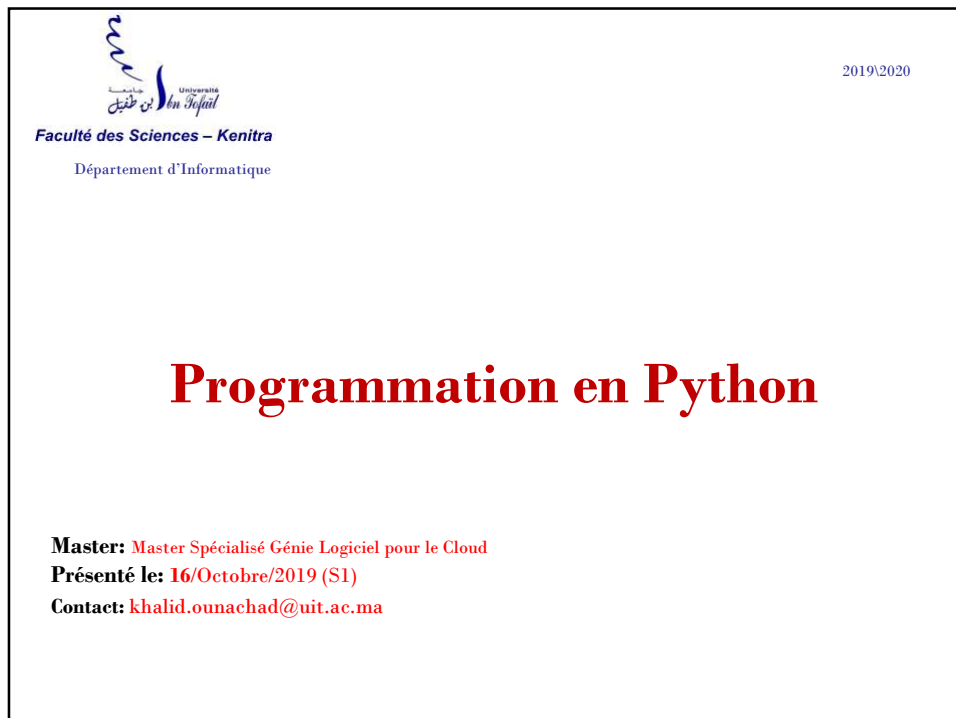




1



2

Les éléments de base en langage Python

- Introduction,
- Types et opérateurs de base ,
- Instructions de base ,
- Choix ,
- Boucles,
- Fonctions.

3

- Introduction,

4

Introduction

- Python est développé depuis 1989 par **Guido van Rossum** et de nombreux contributeurs bénévoles.
- Python est un langage :
 - *gratuit*, et *open source*
 - *portable*,
 - *dynamique*,
 - *extensible*,
 - qui permet (sans l'imposer) une approche modulaire et *orientée objet* de la programmation.
 - Graphique...ect



5

Introduction



- Python 3.x n'est pas une simple amélioration ou extension de Python 2.x.



6

Introduction

Les différentes implémentations:

- **CPython** Implémentation de base basé sur le langage C
ANSI
- **Jython** Implémentation permettant de mixer Python et
java dans la même JVM
- **IronPython** Implémentation permettant d'utiliser
Python pour Microsoft .NET
- **PyPy** Implémentation de Python en Python
- **CLPython** Implémentation de Python en Common Lisp

7

Introduction

Que peut-on faire avec Python ?

- **bases de données**
MySQL, PostgreSQL, Oracle, ...
- **réseaux**
TwistedMatrix, PyRO, ...
- **représentation graphique**
gnuplot, matplotlib, VTK, ...
- **calcul scientifique**
numpy, scipy, sage, ...
- **Traitement d'image:**
open cv, pil
- ...

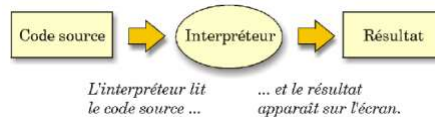
8

Introduction

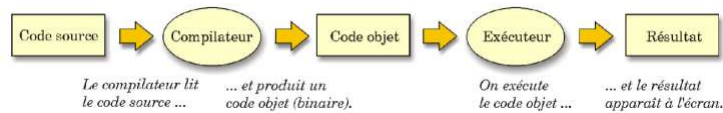
Mon code source ?

Il existe 2 techniques principales pour effectuer la traduction en langage machine de mon code source :

- **Interprétation**



- **Compilation**



Figures tirées du livre « Apprendre à programmer avec Python »

9

Introduction

Mon code source Python ?



Figure tirée du livre « Apprendre à programmer avec Python »

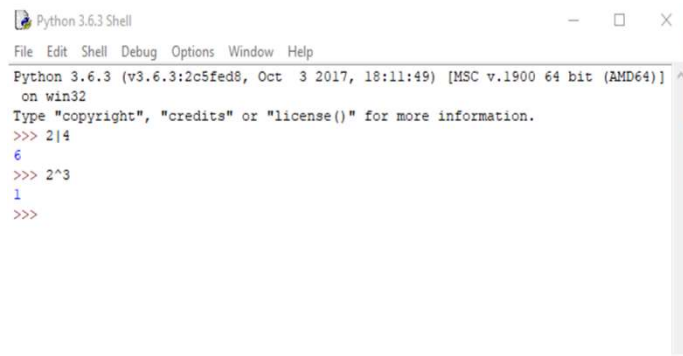
- + interpréteur permettant de tester n'importe quel petit bout(script) de code,
- peut être lent.

10

Introduction

L'interpréteur Python?

Sous Windows :



```
Python 3.6.3 Shell
File Edit Shell Debug Options Window Help
Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2|4
6
>>> 2^3
1
>>>
```

11

- Types et opérateurs de base

12

Types et opérateurs de base

- Mots réservés python:

and as assert break class continue def
del elif else except False finally for from global
if import in is lambda None nonlocal
not or pass raise return True try while
with yield

13

Types et opérateurs de base

- Types de base: simples

Valeur	Type python	exemple
$\in \mathbb{Z}$	int	12,+20,-20
$\in \mathbb{IR}$	float	12., 20.0, 1.23e-6
$\in \mathbb{C}$	complex	2+9j, 2+9J
caractère	Chr	'1','f'
Valeur logique	bool	True ,False

Type permet d'avoir le type ou la
classe du type

14

Types et opérateurs de base

- bases de numérotation et codage(conversions)

En python	exemple	Résultat
bin	bin(2)	0b10
oct	oct(12)	0o14
hex	hex(15)	0xf
int	int(0b10)	2
ord	ord('a')	97(code ASCII..)
chr	chr(97)	'a'

15

Types et opérateurs de base

- Affectation: =

```
>>> # ceci est un commentaire
```

```
>>> i = 30 # i vaut 30
```

```
>>> a, pi = True, 3.141596
```

```
>>> k = r = 2.156
```

16

Types et opérateurs de base

- **Affectation: chaîne de caractères**

```
>>> x = 'hello '

>>> y = "world!"

>>> z = '''hello
... world'''
```

17

Types et opérateurs de base

- **Opérateurs arithmétiques (ou et fonctions)**

opérateur	Syntaxe python
$X + y$	$X + y$
$X - y$	$X - y$
$X \times y$	$X * y$
X / y	X / y
x/y division euclidienne	$X // y$
x^y	$X ** y$, <code>pow(x,y)</code>
$ x $	<code>abs(x)</code>
Reste de la division euclidienne	<code>%</code>

18

Types et opérateurs de base

- Opérateurs abrégés

opérateur	Syntaxe python abrégée
$X = X + y$	$X += y$
$X = X - y$	$X -= y$
$X = X \times y$	$X *= y$
$X = X / y$	$X /= y$
$X = X // y$	$X //= y$
$X = X ** Y$	$X ** = y$
$X = X \% y$	$X \% = y$

19

Types et opérateurs de base

- Opérateurs de comparaisons:

opérateur	Syntaxe python
= (égalité)	$==$
\neq	$!=$
\leq	$<=$
\geq	$>=$
$>$	$>$
$<$	$<$
5 est réel	5 is float
	is not

20

Types et opérateurs de base

- Opérateurs logiques:

opérateur	Syntaxe python
et	and
ou	or
non	not

- Opérateurs d'appartenance:

opérateur	Syntaxe python
€	in
∉	not i

21

Types et opérateurs de base

- Opérateurs binaire(la base binaire):

opérateur	Syntaxe python	Exemple
Et logique	&	>>> 2&3 2
Ou exclusif	^	>>> 2^3 1
Ou logique		>>> 2 3 3
Décalage à gauche	<<	>>> 2<<3 16
Décalage à droit	>>	>>> 2>>3 0

22

- Instructions de base,

23

Instructions de base

- **Lecture:**

- **Syntaxe:**

```
input(prompt=None, /)
```

Read a string from standard input. The trailing newline is stripped.

The prompt string, if given, is printed to standard output without a trailing newline before reading input.

If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Return), raise EOFError. On *nix systems, readline is used if available.

tirée de la documentation de « python3.6.3 »

- **Exemple:**

```
X=float(input(«donner un nombre réel: »))
```

24

Instructions de base

- **Affichage:**

- **Syntaxe:**

```
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

```
Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep: string inserted between values, default a space.
end: string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
```

tirée de la documentation de « python3.6.3 »

- **Exemple:**

```
print(« le langage Python »)
print(2**10)
print(« resultat= »,5/20+30**8)
```

25

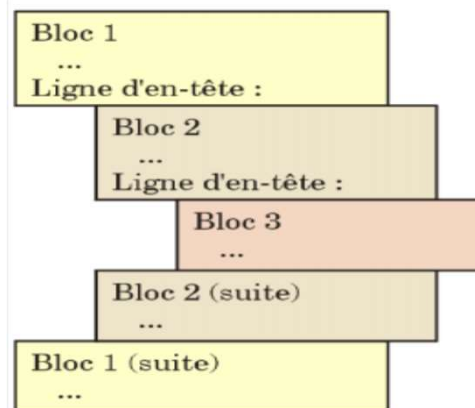
- **Choix(sélections),**

26

Choix

- **Indentation générale**

- **Fonctionnement par bloc**



Figures tirées du livre « Apprendre à programmer avec Python »

27

Choix

- **Simple:**

- **Syntaxe:**

if conditions :

blocs d'instructions

- **Exemple:**

a = -150

if a < 0:

 print ('a est négatif')

28

Choix

- **Double:**

- **Syntaxe:**

if conditions :

blocs d'instructions 1

else:

blocs d'instructions 2

- **Exemple:**

if a < 0:

print ('a est négatif')

else:

print('a est positif ')

29

Choix

- **Multiple:**

- **Syntaxe:**

if conditions1 :

blocs d'instructions 1

elif conditotions2 :

blocs d'instructions 2

elif conditions I :

...

else:

autres instructions

30

Choix

- **Multiple:**

- **Exemple:**

```
a = 10.  
if a > 0:  
    print( 'a est strictement positif' )  
    if a >= 10:  
        print( 'a est un nombre' )  
    else:  
        print( 'a est un chiffre' )  
    a += 1  
elif a is not 0:  
    print( 'a est strictement négatif' )  
else:  
    print( 'a est nul' )
```

31

- **Boucles(Répétitions),**

32

Boucles

- **While:**

- **syntaxe:**

`while` conditions:
instructions

- **Instructions particuliers:**

break : sort de la boucle

continue : remonte au début de la boucle,

pass : ne fait rien,

33

Boucles

- **While:**

- **exemple: y est-il premier ?**

`x = y / 2`

`while x > 1:`

`if y % x == 0:`

`print (y, 'est facteur de', x)`

`break`

`x = x-1`

`print (y, 'est premier')`

34

Boucles

- **For:**

- **Syntaxe:**

```
for cible in séquence d'objets:
    bloc instructions
```

- **range:**

- **Syntaxe**

range(start,stop+1,step)

- **Exemple:**

range(6) donne la séquence(liste) [0,1,2,3,4,5]

range(1,6)----->[1,2,3,4,5]

range(1,6,3)----->[1,4]

35

Boucles

- **For:**

- **Exemple:**

```
prod = 1
for p in range(1, 10):
    prod *= p
    print(prod)
```

- **Exécution:**

```
1
2
6
24
120
720
5040
40320
362880
```

36

Boucles

- **remarque:**

- **zip** : permet de parcourir plusieurs séquences en parallèle
- **map** : applique une méthode sur une ou plusieurs séquences **Exemple:**

```
L1 = [1, 2, 3]
```

```
L2 = [4, 5, 6]
```

```
for (x, y) in zip(L1, L2):
```

```
    print( x, y, '--', x + y)
```

Exécution:

```
1 4 -- 5
```

```
2 5 -- 7
```

```
3 6 -- 9
```

37

- **Fonctions**

38

Fonctions

- Fonction prédéfinies:**

- Syntaxe:

```
from module import fct1,fct2,
```

```
from module import * # pour charger toutes les fonctions du module
```

```
import module
```

Exemple

```
from math import sin,sqrt
```

```
print("la racine carré de « ,x,"est",sqrt(x),"et sin("x,")="sin(x))
```

Ou bien

```
import math
```

```
print(math,sin(x))
```

39

Fonctions

- Quelques fonctions prédéfinies: math**

Fonction	Syntaxe python
$ x $	<code>fabs(x)</code>
\sqrt{x}	<code>sqrt(x)</code>
X^y	<code>pow(x,y)</code>
e^x	<code>exp(x)</code>
\ln	<code>log</code>
\log	<code>log10</code>
Log a base 2	<code>log2</code>
Sin,cos,tg	<code>sin,cos,tan</code>
sinh	<code>Sinh...</code>

40

Fonctions

- **Remarque fonctions prédéfinies:**
- **dir** permet de voir les objets et méthodes disponibles
- **help** permet d'avoir une aide
- **Exemple:**

```
>>> dir(math)
['_doc_', '_loader_', '_name_', '_package_', '_spec_', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e',
'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma',
'log', 'log10', 'logip', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
>>> help(math.log2)
Help on built-in function log2 in module math:

log2(...)
|   log2(x)
|
|   Return the base 2 logarithm of x.
```

41

Fonctions

- **fonctions propres:**
- **Syntaxe:**

```
def NomFonction(arg1, arg2,... argN):
```

```
    ...
```

```
    bloc d'instructions
```

```
    ...
```

```
    return valeurs_resultats
```

42

Fonctions

- **fonctions propres:**

- **Exemple:**

```
def est_premier(n):  
    for i in range(2,n):  
        if n%i==0:  
            return False  
    return True
```

43

Fonctions

- **Fonction réursive:**

- **Exemple:**

```
def f(n):  
    if n==0:  
        return 1  
    else:  
        return n*f(n-1)
```

44

Fonctions

- **Fonction lambda :**

- **Syntaxe:**

lambda argument1,... argumentN : expression utilisant les arguments

Exemple :

```
f = lambda x, i : x**i
```

```
f(2, 4)
```

45

MERCI

46

Travaux Pratiques:

- TP1