Python
Introduction

Nicolas Delestre

INSA
ROUEN NORMANDIE

Plan

1. Caractéristiques
2. Types de base
3. Instructions de base
4. Fonctions
5. Comment utiliser python ?
6. Scripts, module, package

INSA ROUEN NORMANDIE

Historique

- Langage créé en 1990 par Guido van Rossum
  - Branche 2.0 à partir de 2000
  - Branche 3.0 à partir de 2008
- S'inspire de plusieurs langages : ABC, Modula, C, Smalltalk, langages de scripts
- Langage open source (GPL), plusieurs implantations (**CPython**, Pypy, Jython, etc.)
- Versions actuelles (en 2020), deux branches : 2.7 et **3.9** (3.8 utilisé)

INSA ROUEN NORMANDIE

Caractéristiques

- Langage interprété mais aussi compilé (interprété par une machine virtuelle)
- Multi-paradigmes : structuré, orienté objet, fonctionnel, méta programmation (plus d'autres par extension : programmation par aspect - être capable de bien découpler logique métier et services - et programmation logique)
- Typage fort
- Typage dynamique
- *Duck typing* : le type d'un objet est défini par ce qu'il ≪ sait faire ≫ pas par le nom du type
- *Garbage collector*
- Passage de paramètre par référence
- Langage objet (tout est objet : types, fonctions, classes, modules, packages, etc.)
- L'indentation (4 espaces pour la PEP8) marque les blocs
- Sensible à la casse

INSA ROUEN NORMANDIE

Booléen

`bool`

- deux valeurs `True` et `False`
- opérateurs : `or`, `and` et `not`

Pour les tests

Outre les booléens, pour les tests (`if`, `while`), les valeurs suivantes sont équivalentes à `False` : `None`, `0`, `0.0`, `0j`, `''`, `()`, `[]`, `{}`

Opérateurs de comparaison qui retourne un booléen

`<`, `>`, `<=`, `>=`, `==` (égal), `!=`, `is` (identique) et `is not`

INSA ROUEN NORMANDIE

Nombres

Les types numériques

Types de données immuables

- `int` représentant les entiers signés (de 0 à 256 : référence unique), préfixe des constantes (par défaut décimale)
  - `0b` ou `0B` pour binaire
  - `0o` ou `0O` pour octale
  - `0x` ou `0X` pour notation hexadécimale,
- `float` représentant les flottants signés
- `complex` représentant les nombres complexes

Opérateurs et fonctions sur les types numériques

- `+`, `-`, `*`, `/`, `//`, `%`, `**`
- `abs`, `int`, `float`, `complex`, `conjugate` (méthode), `divmod`, `pow`
- `~`, `<<`, `>>`

Sequences 1 / 3

`tuple`

- suite immuable d'objets, qui peuvent être muables, de type divers
- exemple : `1,1.0,"abc",a`, `(1,1.0,"abc",a)` ou `tuple(s)` avec `s` une séquence

`range`

- suite immuable d'entiers
- syntaxe : `range(fin)`, `range(debut, fin[, pas])`
- l'entier de `fin` n'est pas inclu
- exemple : `range(10)`

`list`

- suite muable d'objets de type divers
- exemple : `[1,1.0,"abc",a]` ou `list(s)` avec `s` une séquence

Sequences 2 / 3

slice

- permet de désigner une partie d'une séquence :
- exemples :

```
a=list(range(0,20,2)) # les 10 premiers nombres pairs
>>> a
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
>>> a[:]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
>>> a[0:5] # d'indice >=0 et <5
[0, 2, 4, 6, 8]
>>> a[1:4]
[2, 4, 6]
>>> a[-2:]
[16, 18]
>>> a[0:5:2] # d'indice >=0 et <5 par pas de 2
[0, 4, 8]
```

INSA ROUEN NORMANDIE

Sequences 3 / 3

Opérations, fonctions, méthodes

opérations `in`, `not in`, `+`, `*`, `[i]`, `[i:j]`, `[i:j:k]`

fonctions `len`, `min`, `max`

méthodes `index(X[, i[, i]])`, `count(x)`

Opérations supplémentaires pour les non immuables

opérations `s[i] = x`, `s[i:j] = t`, `s[i:j:k] = t`

fonctions `del s[i:j]`, `del s[i:j:k]`

méthodes `append`, `clear`, `copy`, `extend`, `insert`, `pop`, `remove`, `reverse`

INSA ROUEN NORMANDIE

Ensemble

`set`

- ensemble d'objets *hashables*
- exemple : `{1,2,3,1}` ou `set(s)` avec `s` une séquence

Opérations, fonctions

fonction	méthode	opérateur
		in
		not in
len		
	isdisjoint	
	issubset	<=
		<
	issuperset	>=
		>
	union	|
	intersection	&
	difference	-
	symmetric_difference	^

Dictionnaire

dict

- les clés doivent être *hashables* (pas possible pour les séquences non immuables)
- exemple :

```
>>> d={1:"a",(1,2):"b"}
>>> d[1]
'a'
>>> d[(1,2)]
'b'
```

Opérations, fonctions, méthodes

opérations `in`, `not in`, `[i]`,

fonctions `len`, `iter`,

méthodes `clear`, `copy`, `get`, `items`, `keys`, `pop`, `popitem`, `update`, `values`

Chaîne de caractères

str

- séquence immuable de caractères unicodes
- constantes peuvent utiliser des simples quotes, doubles quotes ou des triples simples quotes ou triples doubles quotes
- de nombreuses méthodes permettant d'interroger, de découper, de retrouver, de remplacer et de formater une chaîne. À chaque fois elles retournent une valeur (par exemple la nouvelle chaîne calculée)
- mise en forme de chaîne avec l'utilisation de l'opérateur % (utilisation d'un tuple si plusieurs valeurs) ou des f-string (à partir de la version 3.6), par exemple :

```
>>> a=1
>>> b="une chaîne"
>>> "a vaut %d et b vaut '%s'" % (a,b)
"a vaut 1 et b vaut 'une chaîne'"
>>> f"a vaut {a} et b vaut {b}"
'a vaut 1 et b vaut une chaine'
```

Affectation

- affectation des références

```
>>> a=12.5
>>> b=12.5
>>> c=a
>>> a is b
False
>>> a is c
True
```

- possibilité de faire plusieurs affectations en une seule fois (utilisation des tuples)

```
>>> a,b=1,2
>>> a,b=b,a
>>> a
2
```

INSA ROUEN NORMANDIE

Conditionnelle

if elif else

- Syntaxe :

```
if condition:
    ...
[elif condition:
    ...
]
[else:
    ...
]
```

Itération déterministe 1 / 2

for in

- Syntaxe :

```
for e in iterable:
    ...
[else:
    ..
]
```

- la partie `else` est exécutée lorsque la séquence ou l'ensemble a été parcouru entièrement (pas exécutée si on sort de la boucle à cause d'un `break`)
- pour l'instant un itérable : séquence, ensemble, dictionnaire (dans ce cas on itère sur les clés)

INSA ROUEN NORMANDIE

Itération déterministe 2 / 2

Fonction enumerate

- Syntaxe :

```
for i,e in enumerate(iterable):
    ...
```

Itération indéterministe

while

- Syntaxe :

```
while condition:
    ...
[else:
    ...
]
```

- la partie `else` est exécutée lorsque la condition devient fausse (pas exécutée si on sort de la boucle à cause d'un `break`

Définition standard de fonction 1 / 3

def

- Syntaxe :

```
def nom(param1[: annotation1], param2[: annot2], ...) [-> annotRetour]:
    """ documentation """
    ...
```

- Il n'y a pas de procédure en python, uniquement des fonctions qui retournent `None` lorsque l'instruction `return` n'est pas utilisée ou utilisée avec aucune valeur
- Les annotations (à partir de python 3.4) sont optionnelles et doivent être des expressions python (souvent les types attendus, cela peut poser problème pour les classes en cours de définition)
- Le passage de paramètre est un passage par référence : possibilité de changer l'état de l'objet (si muable) mais pas possible de changer d'objet
- Toutes les variables utilisées dans une fonction sont des variables locales (sauf utilisation explicite du mot clé `global`)
- Lors de l'appel d'une fonction, l'association paramètres effectifs paramètres formels peut être « positionnel » ou « nommé »

Définition standard de fonction 2 / 3

Exemple

```
def est_premier(n):
    """permet de savoir si un nombre est premier ou pas"""
    if n % 2 == 0:
        return False
    else:
        for i in range(3, n // 2, 2):
            if n % i == 0:
                return False
    return True
```

Définition standard de fonction 3 / 3

Exemple

```
def est_premier(n: int) -> bool:
    """permet de savoir si un nombre est premier ou pas"""
    if n % 2 == 0:
        return False
    else:
        for i in range(3, n // 2, 2):
            if n % i == 0:
                return False
    return True
```

Définition de fonction d'arité variable 1 / 6

En utilisant des valeurs par défaut

- Syntaxe :

```
def nom(param1, ..., paramOpt1 = val1, paramOpt2 = val2...):
    """ documentation """
    ...
```

- les paramètres formels ayant une valeur par défaut deviennent optionnels
- dès qu'un paramètre formel à une valeur par défaut, les suivants doivent en avoir une aussi

INSA ROUEN NORMANDIE

Définition de fonction d'arité variable 2 / 6

Exemple

```
def somme_naturels(fin, debut=1, pas=1):
    res = 0
    for i in range(debut,fin,pas):
        res = res + i
    return res
```

```
def somme_naturels(fin: int, debut: int=1, pas: int=1) -> int:
    res = 0
    for i in range(debut,fin,pas):
        res = res + i
    return res
```

```
>>> somme_naturels(10)
45
>>> somme_naturels(10,5)
35
>>> somme_naturels(10,pas=2)
25
>>> somme_naturels(debut=1,fin=10,pas=2)
25
```

Définition de fonction d'arité variable 3 / 6

En utilisant *args

- Syntaxe :

```
def nom(param1, ..., *args):
    """ documentation """
    ...
```

- Cela signifie que le nombre de paramètres effectifs non nommés peut être en nombre variable
- On peut transformer une séquence en une suite variable d'arguments grâce à l'opérateur *
- *args doit être déclaré après les paramètres formels nommés
- Lors de l'appel, il doit y avoir une valeur pour tous les paramètres nommés
- args est un séquence

Définition de fonction d'arité variable 4 / 6

Exemple

```
def somme_nombres(debut, fin, *args):
    res = 0
    for i in args[debut:fin]:
        res = res + i
    return res
```

```
>>> somme_nombres(0,3,10,20,30,40,50)
60
>>> somme_nombres(0,3,10,20,30,40,50,60,70)
60
>>> somme_nombres(0,3,*[10,20,30,40,50,60,70])
60
>>> somme_nombres(0,7,10,20,30,40,50,60,70)
280
```

INSA ROUEN NORMANDIE

Définition de fonction d'arité variable 5 / 6

En utilisant **kargs

- Syntaxe :

```
def nom(param1, ...,  **kargs):
    """ documentation """
    ...
```

- Cela signifie que le nombre de paramètres effectifs nommés peut être en nombre variable (différents des paramètres formels)
- `kargs` est un dictionnaire
- `**kargs` doit être déclaré après `*args`
- On peut transformer un dictionnaire ayant des clés de type `str` en une suite de paramètres effectifs nommés grâce à l'opérateur `**`

INSA ROUEN NORMANDIE

Définition de fonction d'arité variable 6 / 6

Exemple

```
def valeur_d_une_cle(dic, cle, valeur_si_non_present):
    if dic:
        if cle in dic.keys():
            return dic[cle]
        else:
            return valeur_si_non_present
    else:
        return valeur_si_non_present

def somme_nombres_v2(*args, **kargs):
    debut = valeur_d_une_cle(kargs,'debut',0)
    fin = valeur_d_une_cle(kargs,'fin',len(args))
    res = 0
    for i in args[debut:fin]:
        res = res + i
    return res
```

```
>>> somme_nombres_v2(10,20,30,40,debut=1)
90
>>> somme_nombres_v2(10,20,30,40,debut=1,fin=3)
50
>>> somme_nombres_v2(10,20,30,40,fin=3)
60
>>> somme_nombres_v2(10,20,30,40,50,60,70,debut=1,fin=3)
50
```

Instruction pass

- Permet de définir une fonction qui ne fait rien (on reporte à plus tard son développement)

Exemple

```
def fonction_qui_ne_fait_rien():
    pass
```

fonctions de base 1 / 2

`print`

- Fonction qui affiche sur la sortie standard une représentation sous forme de chaîne des objets (appel de la fonction `str`)
- Plusieurs paramètres nommés : `sep`, `end`, `file=sys.stdout`, `flush=False`

`input`

- Fonction qui récupère de l'entrée standard une chaîne de caractères
- un seul paramètre optionnel : le prompt

INSA ROUEN NORMANDIE

fonctions de base 2 / 2

https://docs.python.org/3/library/functions.html

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()		

INSA ROUEN NORMANDIE

Comme environnement interactif

python3

- interpréteur de base, interactions limitées
- CTRL+D pour sortir

ipython3

- ipython3 : interpréteur étendu
  - complétion
  - “commandes magiques”, commencent par %, par exemple : %paste, %autoreload (commande étendu : %load_ext
  - Accès facilité à numpy et matplotlib
- CTRL+D pour sortir

INSA ROUEN NORMANDIE

Comme script exécutable

python3

- `python3 [options] [script.py | -c cmd] [args]`
- Sous UNIX (Linux, MacOS) comme exécutable : le début d'un script doit déclarer, sous forme de commentaire, le programme à utiliser pour exécuter le script (sous unix avec les droits en exécution)

```
#!/usr/bin/env python3
```

Environnement de développement

- emacs
- spyder : environnement plutôt dédié aux développements scientifiques
- eclipse (pydev)
- Eric

INSA ROUEN NORMANDIE

Organisation d'un .py

1. Déclaration de l'encodage du fichier (obligatoirement utf-8 en python3, donc optionnel)

   ```
   # -*- coding: utf-8 -*-
   ```

2. Le rôle du fichier sous forme de *docstring* qui sera utilisé par help()
3. Des métadonnées qui seront utilisées par `help()` : `__author__` (str), `__contact__` (sequence de `str`), `__credits__` (sequence de `str`), `__maintainer__` (`str`) `__email__` (`str`) `__version__` (`str`), `__copyright__` (`str`), `__date__` (`str` au format ISO 8601)
4. Les déclarations :
   - Variables globales (en majuscule)
   - Classes
   - Fonctions
   - Le code à exécuter (équivalent du *main* dans d'autres langages)

ROUEN NORMANDIE

Script

- Suite de déclarations et d'instructions enregistrées dans un fichier pour pouvoir être interprétées par le programme python
- Un script est donc considéré comme un programme

INSA ROUEN NORMANDIE

Module 1 / 4

- Suite de déclarations et d'instructions enregistrées dans un fichier qui peuvent être importées (dans un autre module ou dans un script)
- Les déclarations sont propres au module
- Les instructions sont exécutées au premier `import` (elles servent à initialiser certaines variables globales au module)
- L'import est réalisé grâce :
  - à l'instruction `import` (l'utilisation d'une déclaration nécessite de préfixer la déclaration par le nom du module)
  - à l'instruction `from .. import ..` (l'utilisation d'une déclaration est alors direct)
    - l'utilisation de `*` permet d'importer toutes des déclarations d'un module (à éviter)
  - l'utilisation de `as` permet de renommer localement un module ou une déclaration d'un module
- Un module peut être utilisé en tant que script.
  `if __name__== "__main__"`: permet de conditionner l'interprétation d'instructions dans ce cas

Module 2 / 4

- Lorsqu'un module est importé, python recherche le fichier correspondant dans la liste des répertoires référencés par la variable `sys.path`
- `sys.path` contient
  1. le répertoire courant
  2. les répertoires présents dans la variable d'environnement `PYTHONPATH`
  3. les répertoires systèmes python

INSA ROUEN NORMANDIE

Module 3 / 4

fibo.py

```
#!/usr/bin/env python3
"""
Exemple issu du tutoriel de python: https://docs.python.org/
"""

__author__ = "Nicolas Delestre"
__contact__ = "Nicolas Delestre"
__copyright__ = "Copyleft"
__credits__ = ["Nicolas Delestre"]
__license__ = "GPL"
__version__ = "1.0.1"
__maintainer__ = "Nicolas Delestre"
__email__ = "nicolas.delestre@insa-rouen.fr"
__status__ = "Production"

def fib(n):
    """ retourne les nombres de fibonacci jusqu'au rang n """
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result

if __name__ == "__main__":
    import sys
    print(fib(int(sys.argv[1])))
```

Module 4 / 4

```
$ python3 fibo.py 10
[1, 1, 2, 3, 5, 8]
$ chmod +x fibo.py
$ ./fibo.py 10
[1, 1, 2, 3, 5, 8]
$ python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import fibo
>>> fibo.fib(10)
[1, 1, 2, 3, 5, 8]
>>> import fibo as fb
>>> fb.fib(10)
[1, 1, 2, 3, 5, 8]
>>> from fibo import fib as fibonacci
>>> fibonacci(10)
[1, 1, 2, 3, 5, 8]
```

INSA ROUEN NORMANDIE

Package 1 / 3

- Organisation hiérarchique de packages et de modules
- Les packages sont représentés par :
  - un répertoire, le contenu du package (package ou module) est dans le répertoire
  - un fichier __init__.py (obligatoire) dans le répertoire, qui contient une suite d'instructions qui permet d'initialiser le package (exécutée lors du premier import).
    C'est ici qu'est contrôlé le `import *` pour un package (initialisation de la variable `__all__` du package)

INSA ROUEN NORMANDIE

Package 2 / 3

Exemple issu du tutoriel python

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

INSA ROUEN NORMANDIE

Package 3 / 3

Exemple issu du tutoriel python

```
>>> import sound.effects.echo
>>> sound.effects.echo.echofilter(input, output, delay=0.7, atten=4)
...
>>> from sound.effects import echo
>>> echo.echofilter(input, output, delay=0.7, atten=4)
...
>>> from sound.effects.echo import echofilter
>>> echofilter(input, output, delay=0.7, atten=4)
```

INSA ROUEN NORMANDIE