

Bienvenue au Python!

Nguyễn Kim Thắng

kimthang.nguyen@univ-evry.fr

Mails: [L3INF]

bureau 209, IBGBI
IBISC, Univ. Evry, University Paris-Saclay

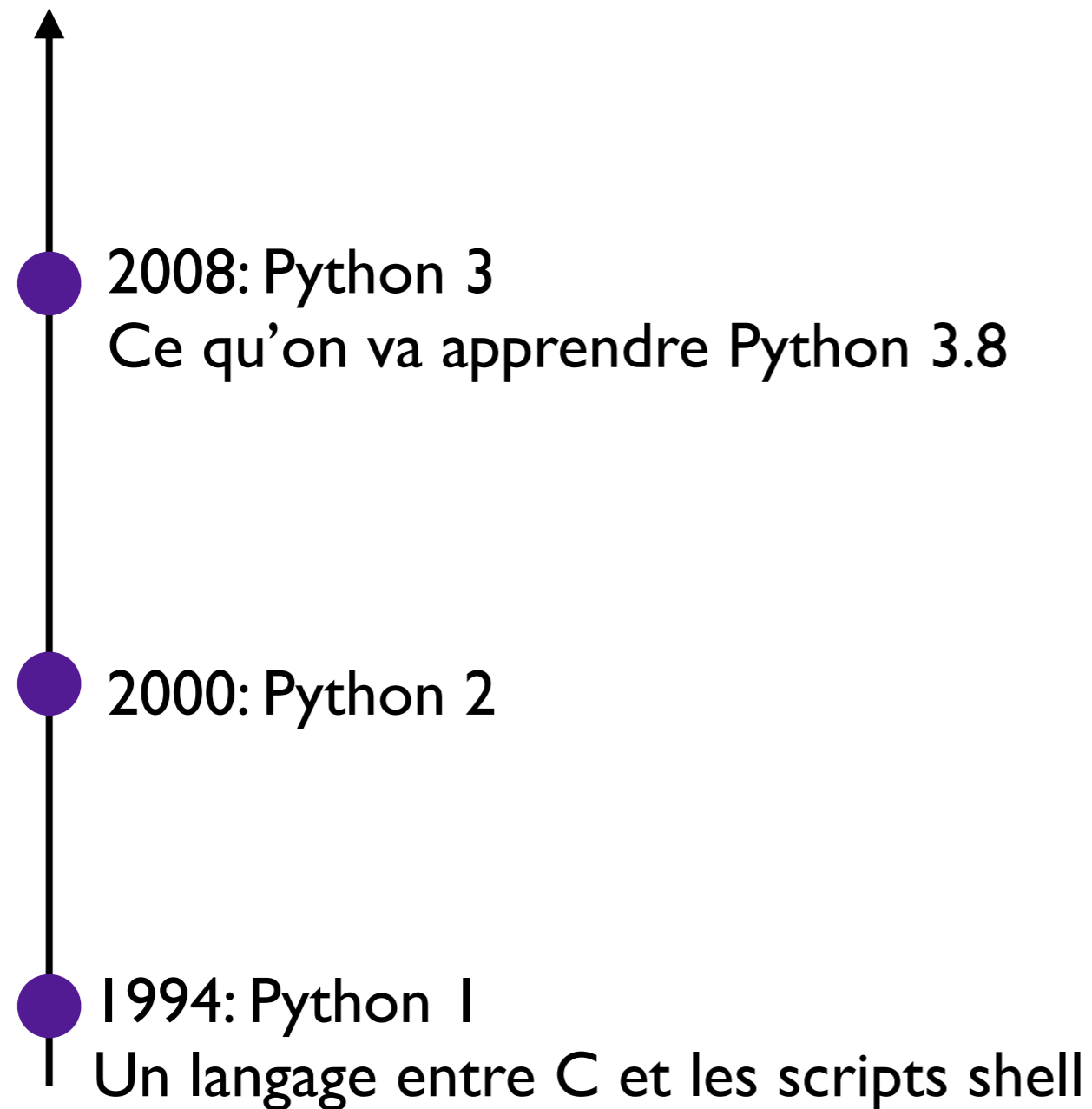
Bienvenue au Python

- Objectifs du cours
- Qu'est-ce Python?
- Pourquoi Python?
- Déroulement du cours
- Premiers pas en Python

Objectif du cours

- Fondamentals du Python
- Ecrire de “bon” programmes de Python
- Utiliser Python pour différentes tâches, ... en particulier en Apprentissage et Science des Données
- Comprendre la puissance (et la limite) de Python

Qu'est-ce Python?



Pourquoi Python

- Python aide à trouver du (bon) travail?



- Avoir de bonnes notes

>>> import this

The Zen of Python, author: Tim Peters

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.

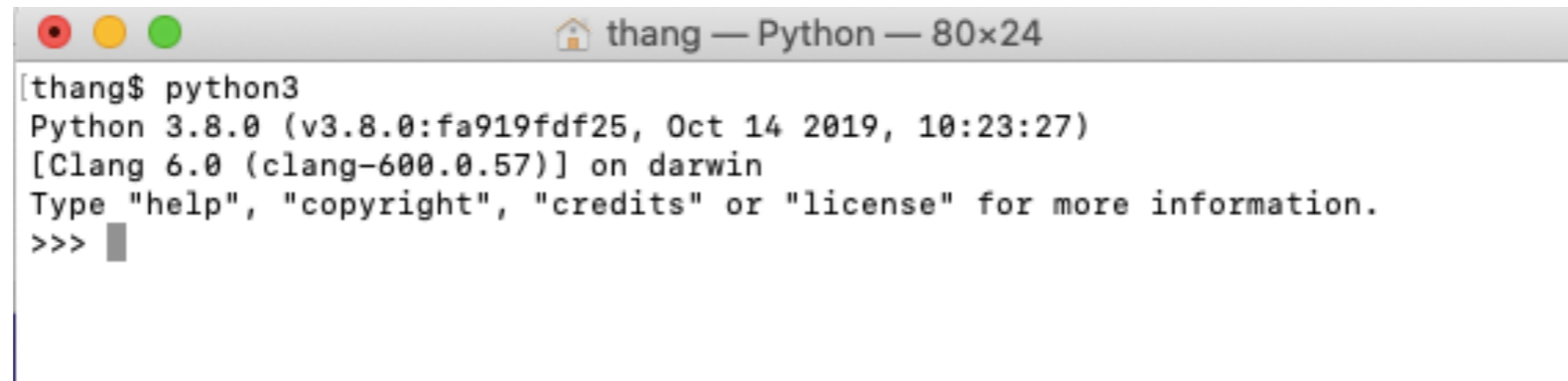
>>> import this

- Special cases aren't special enough to break the rules.
- Although **practicality beats purity**.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, **refuse the temptation to guess**.
- There should be one--and preferably only one--obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than **right** now.
- If the implementation is **hard to explain, it's a bad idea**.
- If the implementation is **easy to explain, it may be a good idea**.
- Namespaces are one honking great idea -- let's do more of those!

Contenu d'aujourd'hui

- Interprétation interactive
- Commentaires
- Variables et Types
- Nombres et Booléens
- Chaînes de caractères et listes
- Input/Output (consoles)
- Control flow (if/else)
- Boucles
- Fonctions
- Expressions

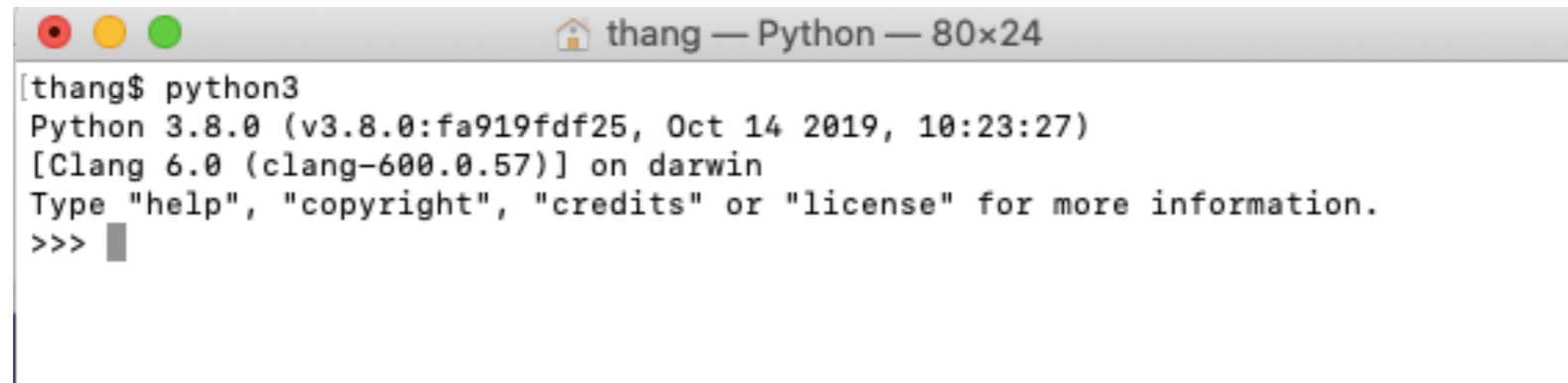
Interprétation interactive



```
thang — Python — 80x24
[thang$ python3
Python 3.8.0 (v3.8.0:fa919fdf25, Oct 14 2019, 10:23:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

- Pas de compilateur, pas de fichier exécutable.
- L'interprète Python analyse le fichier .py et interprète le code au fur et à mesure.
- Il n'y a pas de compilation préalable du code source en code machine.
- Sandbox pour expérimenter Python
- Raccourcir le cycle de code-débogage

Interprétation interactive

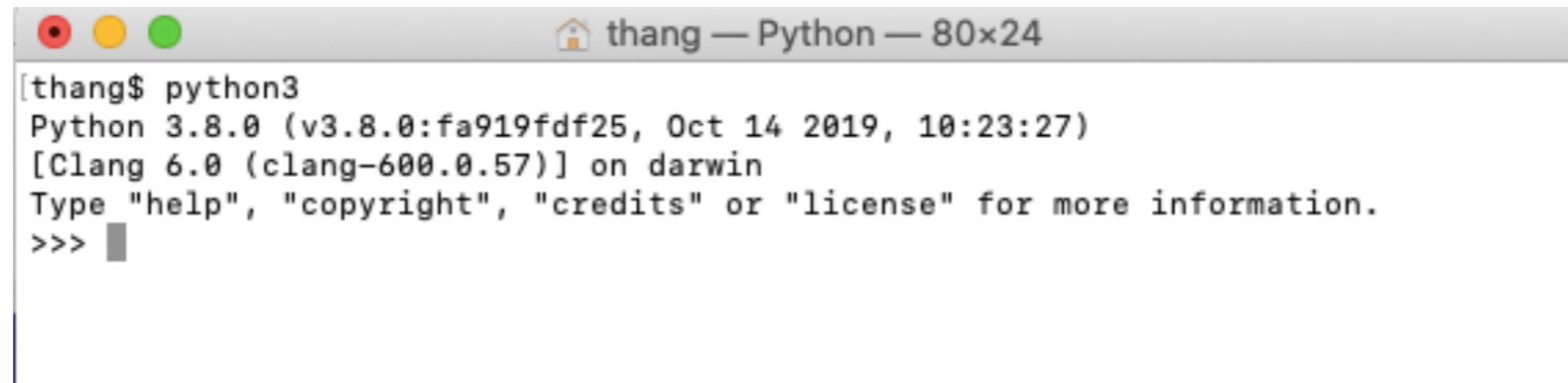
A screenshot of a terminal window titled 'thang — Python — 80x24'. The terminal shows the command '[thang\$ python3]' followed by the output: 'Python 3.8.0 (v3.8.0:fa919fdf25, Oct 14 2019, 10:23:27)', '[Clang 6.0 (clang-600.0.57)] on darwin', and 'Type "help", "copyright", "credits" or "license" for more information.'. The prompt '>>>' is visible at the end of the output.

```
[thang$ python3  
Python 3.8.0 (v3.8.0:fa919fdf25, Oct 14 2019, 10:23:27)  
[Clang 6.0 (clang-600.0.57)] on darwin  
Type "help", "copyright", "credits" or "license" for more information.  
>>> █
```

□ Python: langage interprété

- Pas de compilateur, pas de fichier exécutable.
- L'interprète Python analyse le fichier .py et interprète le code au fur et à mesure.
- Il n'y a pas de compilation préalable du code source en code machine.
- Sandbox pour expérimenter Python
- Raccourcir le cycle de code-débogage

Interprétation interactive



```
thang — Python — 80x24
[thang$ python3
Python 3.8.0 (v3.8.0:fa919fdf25, Oct 14 2019, 10:23:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

- **Python**: langage interprété
 - Pas de compilateur, pas de fichier exécutable.
 - L'interprète Python analyse le fichier .py et interprète le code au fur et à mesure.
 - Il n'y a pas de compilation préalable du code source en code machine.
- Fonctionnalités et interprète interactif
 - Sandbox pour expérimenter Python
 - Raccourcir le cycle de code-débogage

Commentaires

Commenter une ligne en Python avec le symbole

“ “ “

Commentaire sur plusieurs lignes
entre les guillemets

“ “ “

Variables

`int x = 2;` # Java, C, C++

`x = 2` # Python: pas de type, pas de ;

- Les variables en Python sont typés dynamiquement. Elles prennent en charge le type de l'objet qu'elles représentent.

```
[thang$ python3
Python 3.8.0 (v3.8.0:fa919fdf25, Oct 14 2019, 10:23:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
[>>> x = 2
[>>> type(x)
<class 'int'>
>>> █
```

```
[thang$ python3
Python 3.8.0 (v3.8.0:fa919fdf25, Oct 14 2019, 10:23:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
[>>> x = 2
[>>> type(x)
<class 'int'>
[>>> x = "cours de Python"
[>>> type(x)
<class 'str'>
>>> █
```

Nombres

- Python a (seulement) trois types numériques: float, int et complex

5 # int

5.0 # float

2 + 3 # = 5 (int)

2.3 + 4 # = 6.3 (float)

2 * 4 # = 8 (int)

2 / 3 # = 0.6666 (float)

7 // 3 # = 2 (int; division entière)

7 % 3 # = 1 (int; modulo)

3 ** 3 # = 27 (int; opérateur exponentiel)

Nombres

- L'ajout d'un `=` précédé d'une opération permet d'exécuter l'opération, puis affecter la nouvelle valeur à une variable.

`x += 5` `# x = x + 5`

`x -= 5` `# x = x - 5`

`x *= 5` `# x = x * 5`

`x /= 5` `# x = x / 5`

`x //= 5` `# x = x // 5`

`x %= 5` `# x = x % 5`

`x **= 5` `# x = x ** 5`

Booléens

- Type booléenne est une sous-type de int: `False == 0, True == 1`

True
False

<code>not True</code>	<code># False</code>
<code>True and False</code>	<code># False</code>
<code>True or False</code>	<code># True</code>

<code>2 == 2</code>	<code># True</code>
<code>2 != 3</code>	<code># True</code>
<code>4 > 5</code>	<code># False</code>
<code>9 >= 6</code>	<code># True</code>
<code>5 > 4 > 3</code>	<code># True (5 > 4 and 4 > 3)</code>

Chaînes de caractères

- Python n'a pas de type explicite char. Par contre, char est une chaîne de caractère de longueur une.
- Les deux “...” ou ‘...’ représentent des chaînes de caractères
- Les chaînes de caractères en Python sont en Unicode (donc, on peut utiliser des émojis)

```
[>>> y = "J'aime 🥰 Python"  
[>>> y + '!'  
"J'aime 🥰 Python!"  
>>> █
```

- Concaténation: chaînes de caractères et chaînes de caractères (seulement)
- Si y est un entier, il faut convertir en chaînes de caractère str(y) avant la concaténation.

Indices des chaînes de caractères

cours = “

0	1	2	3	4	5	6	7	8	9	10	11
h	a	p	.	p	y			c	o	d	e

”

cours[début: fin: pas]

inclu

non-inclu

Indices des chaînes de caractères

cours = “

0	1	2	3	4	5	6	7	8	9	10	11
h	a	p	.	p	y			c	o	d	e

”

cours[début: fin: pas]

inclu

non-inclu

cours[2] p # élément à position 2

cours[:3] hap # éléments du début jusqu'à la position 3 non-incluse

cours[5:] y code # éléments de la position 5 incluse jusqu'à la fin

Indices des chaînes de caractères

cours = “

0	1	2	3	4	5	6	7	8	9	10	11
h	a	p	.	p	y		c	o	d	e	”
-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	0

cours[début: fin: pas]

inclu

non-inclu

cours[-2] d # élément 2e de la fin

cours[1:8:2] a.yc # de pos 1 à 8 avec les pas de longueur 2

cours[8:1:-2] o pp # de pos 8 à 1 avec les pas de longueur -2

Comment afficher le string à l'envers ?

cours[::-1]

Lists — ArrayList/Vectors en Python

```
# créer une liste
```

```
empty = []
```

```
letters = ["a", "b", "c"]
```

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

```
# Une liste peut contenir différentes types
```

```
many_types = ["a", 2, 3, [4, 5]]
```

```
# petite question many_types[3] ?
```

```
many_types[3] = [4, 5]
```

```
many_types[3][1] = 5
```

```
# utiliser la fonction "extend" pour ajouter les éléments dans la liste
```

```
numbers.extend([4, 5, 6])
```

```
number = [1, 2, 3, 4, 5, 6]
```

Lists — ArrayList/Vectors en Python

```
numbers = [1, 2, 3, 4]
```

```
# les indices suivent de mêmes règles comme string
```

```
numbers[1:-1] = [2, 3]
```

```
numbers[::2] = [1, 3]
```

```
numbers[1:3:-1] = [] # qu'est-ce qui se passe ?
```

```
numbers[3:1:-1] = [4, 3] # plutôt comme ça
```

Enquêtes sur les collections

longueur

`len([]) = 0`

`len("Python") = 6`

`len([1, 2, 3, "Python"]) = 4`

membre

`0 in [1, 2, 3] = False`

`"3" in [1, 2, 3, "Python"] = False` # chaînes vs int

`"Py" in "Python" = True`

Input/Output

```
# lire l'entrée et sauvegarder dans une variable  
>>> cours = input("Quel est votre cours préféré?")  
  
>>> print ("J'aime bien " + cours)
```


Control Flow

```
if on_apprend_bien :  
    print (“On aura de bonnes notes”)
```

- Les parenthèses ne sont pas nécessaires autour de la condition
- ‘:’ à la fin de la condition
- utiliser 4 espaces (une tab) pour l’indentation

Control Flow

```
if on_apprend_bien :  
    print (“On aura de bonnes notes”)  
elseif on_n’apprend_pas_bien :  
    print (“On n’aura pas de bonnes notes”)  
else :  
    print (“On ne sait pas”)
```

- `elseif` et `else` sont optionnels
- Python n’a pas de `switch`, à la place utiliser `if/elseif/else`

Control Flow

- Lorsque le code ne fonctionne pas au moment de l'exécution, il retourne **Exception**
- Lorsque la syntaxe n'est pas correcte, il retourne **SyntaxError**

```
n = int(input("Combien étudiants dans la classe? "))  
  
# Combien étudiants dans la classe? 20  
# Combien étudiants dans la classe? Vingt  
# => Raises ValueError (un type de Exception)
```

- Solution

```
while True:  
    try  
        n = int(input("Combien étudiants dans la classe? "))  
        break  
    except ValueError  
        print("Veuillez entrer un nombre ")
```

Control Flow

```
try:  
    code_dangereux()  
except Quelque_Erreur:  
    code_alternative_1()  
except Autre_Erreur:  
    code_alternative_2()
```

Boucles for

```
for element in ensemble  
    process (item)
```

- Il n'y a pas de compteur dans la boucle. “element” prends les valeurs séquentielle dans “ensemble”
- “ensemble” peut être une chaîne de caractères, une liste, etc (on va voir plus en détail)

Boucles for

```
>>> for ch in "L3ASR":  
    print (ch)
```

```
L  
3  
A  
S  
R
```

```
>>> for num in [1, 2, 3, 4]:  
    print (num**2)
```

```
1  
4  
9  
16
```

Range

- La fonction `range` génère un itérable sur une plage de nombres
- Syntaxe: `range(stop)` ou `range(start, stop, step)` (comme sur les chaînes de caractères)

```
>>> range(3)          # Générer 0, 1, 2
```

```
>>> range(5, 10)     # Générer 5, 6, 7, 8, 9
```

```
>>> range(2, 12, 3)  # Générer 2, 5, 8, 11
```

```
>>> range(-7, -30, -5) # Générer -7, -12, -17, -22, -27
```

Break et Continue

```
>>> for n in range(2, 10) :  
    if n == 6 :  
        break  
    print(n)
```

2
3
4
5

```
>>> for letter in "L3ASR"  
    if letter in "L3"  
        continue  
    print(letter)
```

A
S
R

Boucles while

`while` condition :
faire()

- Les syntaxes sont similaires à `if`

```
>>> n = 1
>>> while n < 10 :
    print(n)
    n *= 3
1
3
9
```

Fonctions

```
def nom_fonction(param1, param2):  
    valeur = faire_qqc()  
    return valeur
```

- Le mot clé `def` définit une fonction
- Les paramètres n'ont pas de type explicite (pas comme C, C++, Java)
- `return` est optional. S'il n'y a pas de `return`, ça retourne implicitement `None`

Un exemple de fonction

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

Expressions d'affections

- Nouvelle fonctionnalité de Python 3.8
- Utiliser `:=` pour affecter des expressions aux variables

Ancien

```
commande = input("Que voulez vous commander?")
while commande != "Rien" :
    print ("Vous avez commandé:" + commande)
    commande = input("Que voulez vous commander?")
```

Nouveau

```
while (commande := input("Que voulez vous commander?")) != "Rien" :
    print ("Vous avez commandé:" + commande)
```

- Clarté compte (ne pas utiliser si cela compromise la clarté)