Python pour les scientifiques
Partie I – Les objets

Cyril Desjouy

June, 2016
Updated : March 26, 2019

Plan

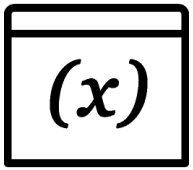
Les objets sous Python

Photography by Joe Lodge

Qu'est ce qu'un `objet` sous Python ?

- Un *objet* est un morceau de programme contenant des données
- A chaque objet peut être associé un certain nombre d'**attributs** et de fonctions spécialisées appelées **méthodes** qui agissent sur les objets
- Chaque objet est défini par son **identité**, son **type**, et sa **valeur**

Identité	• Ne change jamais une fois que l'objet a été crée • Représente "l'adresse" en mémoire • L'opérateur `is` compare l'identité de deux objets • La fonction `id`() retourne un entier représentant son identité
Type	• Ne change jamais une fois que l'objet a été crée • Détermine les opérations que l'objet supporte • Détermine les attributs et méthodes héritées par l'objet • La fonction `type`() retourne le type d'un objet
Valeur	• Peut changer, mais pas toujours ! • Les objets dont la valeur peut changer sont appelés **muable** • Les objets dont la valeur ne peut pas changer sont appelés **immuable** • La **mutabilité** d'un objet est déterminée par son type



Comment déclarer une variable sous Python ?

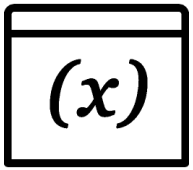
- Caractère "=" pour référencer un objet
- Type assigné dynamiquement

Interpréteur

```
In  : a = 1 ↵
```

Espace des variables

Espace des objets



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement

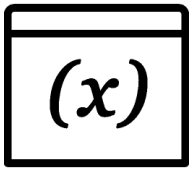
Interpréteur

```
In  : a = 1 ↵
```

Espace des variables

Espace des objets

ref	0	
type	int	
id	0001	1
meth	...	
attr	...	



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement

Interpréteur

```
In  : a = 1 ↵
```

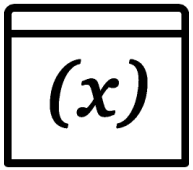
Espace des variables

a

Espace des objets

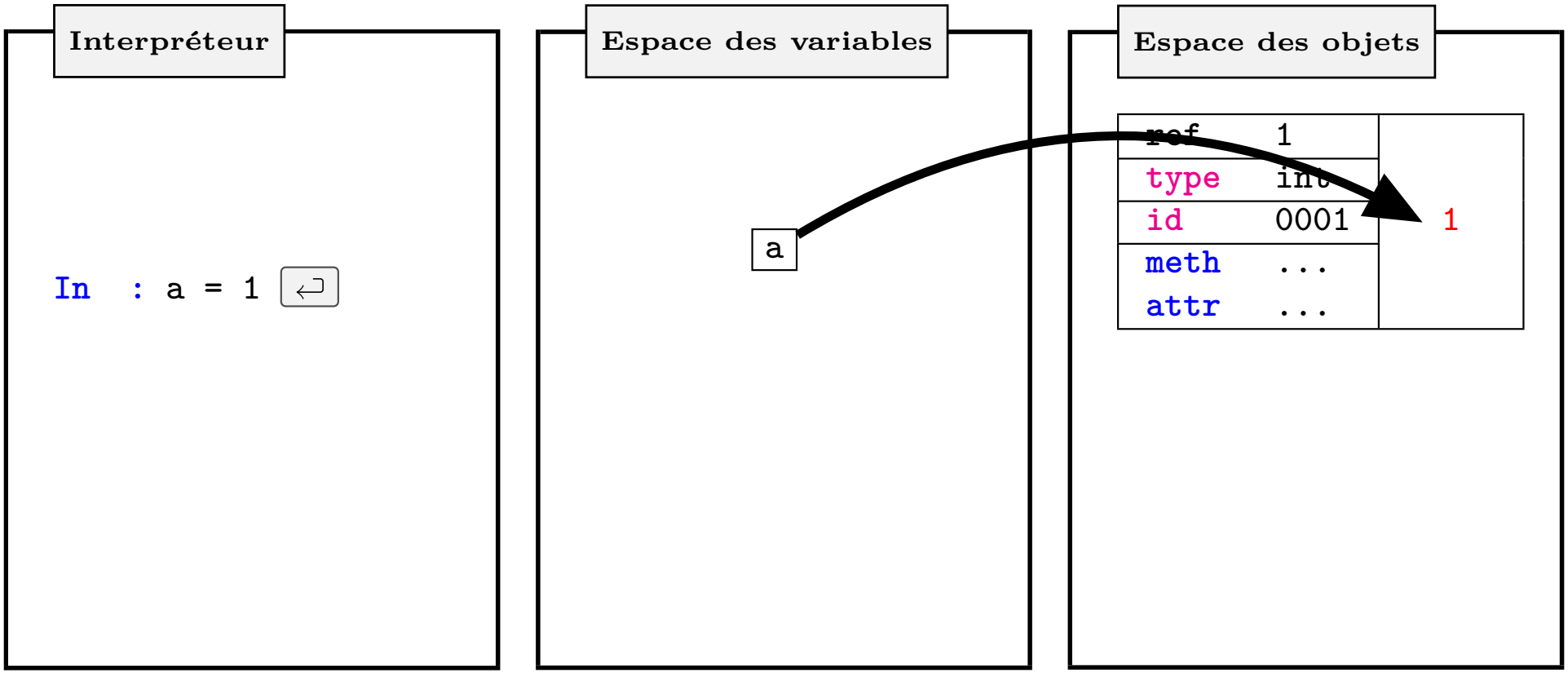
ref	0
type	int
id	0001
meth	...
attr	...

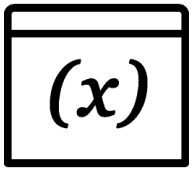
1



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement





Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement

Interpréteur

```
In  : a = 1 ↵
In  : b = 1 ↵
```

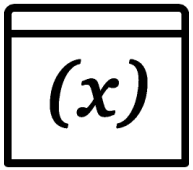
Espace des variables

a

Espace des objets

ref	1
type	int
id	0001
meth	...
attr	...

1



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement

Interpréteur

```
In  : a = 1 ↵
In  : b = 1 ↵
```

Espace des variables

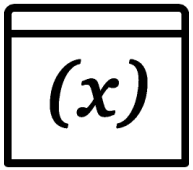
a

b

Espace des objets

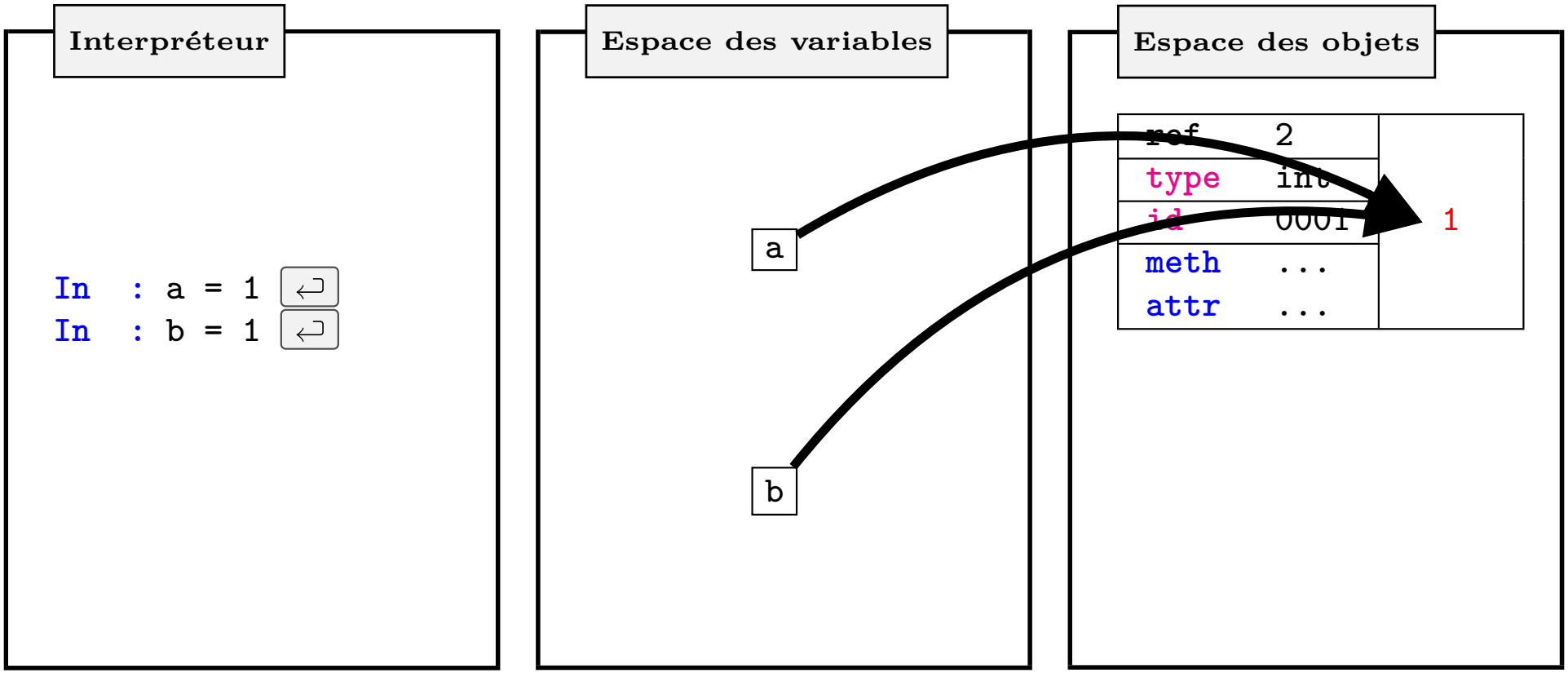
ref	1
type	int
id	0001
meth	...
attr	...

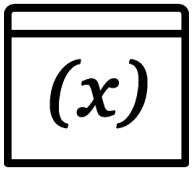
1



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement





Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement

Interpréteur

```
In  : a = 1 ↵
In  : b = 1 ↵
In  : b = "abc" ↵
```

Espace des variables

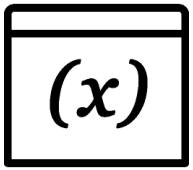
a

b

Espace des objets

ref	2
type	int
id	0001
meth	...
attr	...

1



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement

Interpréteur

```
In  : a = 1 ↵
In  : b = 1 ↵
In  : b = "abc" ↵
```

Espace des variables

a

b

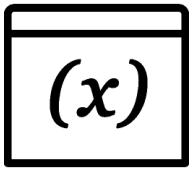
Espace des objets

ref	2
type	int
id	0001
meth	...
attr	...

1

ref	0
type	str
id	0108
meth	...
attr	...

"abc"



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement

Interpréteur

```
In  : a = 1 ↵
In  : b = 1 ↵
In  : b = "abc" ↵
```

Espace des variables

a

b

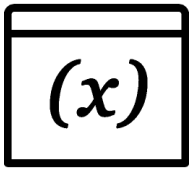
Espace des objets

ref	1
type	int
id	0001
meth	...
attr	...

1

ref	0
type	str
id	0108
meth	...
attr	...

"abc"



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement

Interpréteur

```
In  : a = 1 ↵
In  : b = 1 ↵
In  : b = "abc" ↵
```

Espace des variables

a

b

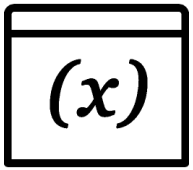
Espace des objets

ref	1
type	int
id	0001
meth	...
attr	...

1

ref	1
type	str
id	0108
meth	...
attr	...

"abc"



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement

Interpréteur

```
In  : a = 1 ↵
In  : b = 1 ↵
In  : b = "abc" ↵
In  : a = b ↵
```

Espace des variables

a

b

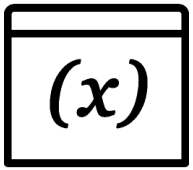
Espace des objets

ref	1
type	int
id	0001
meth	...
attr	...

1

ref	1
type	str
id	0108
meth	...
attr	...

"abc"



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement

Interpréteur

```
In  : a = 1 ↵
In  : b = 1 ↵
In  : b = "abc" ↵
In  : a = b ↵
```

Espace des variables

a

b

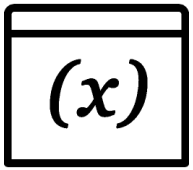
Espace des objets

ref	0
type	int
id	0001
meth	...
attr	...

1

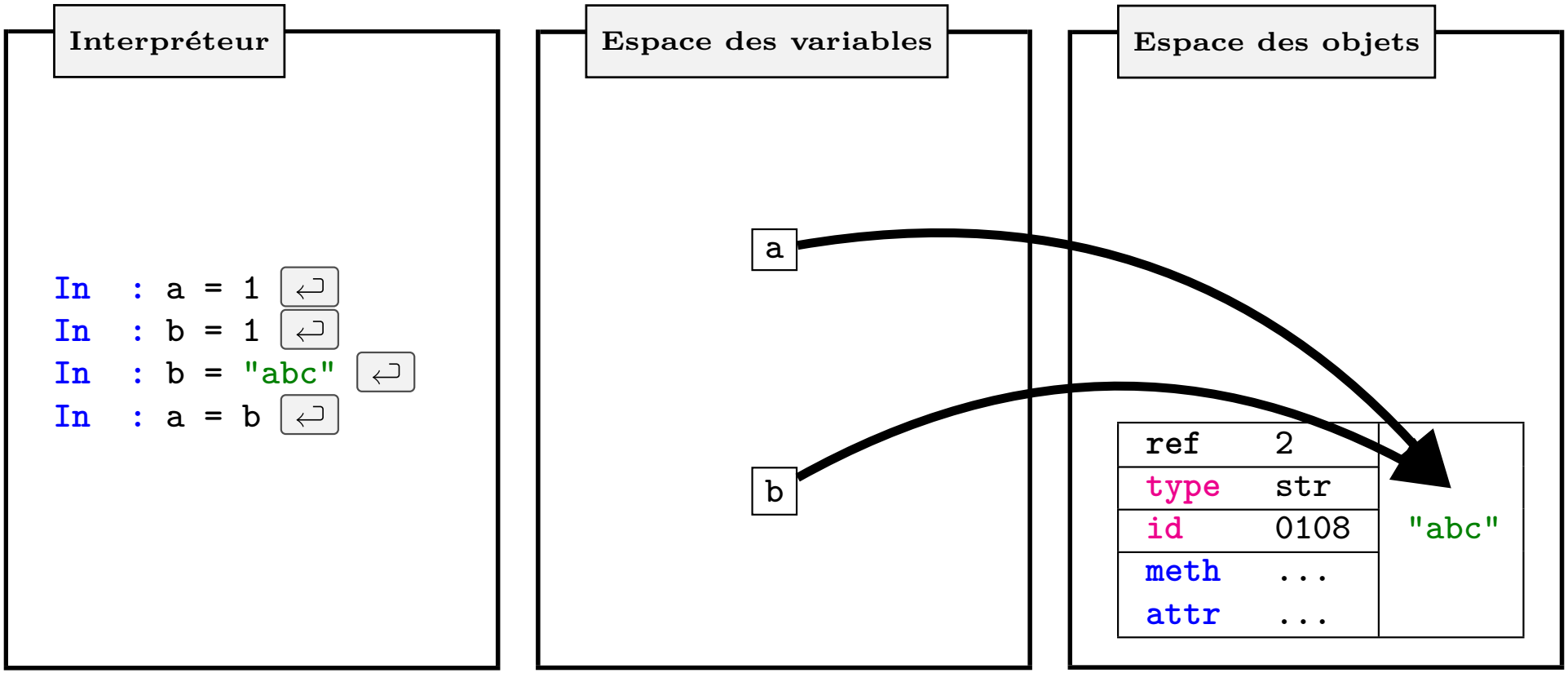
ref	1
type	str
id	0108
meth	...
attr	...

"abc"



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement
- Lorsqu'un objet n'est plus référencé : le "***Garbage collector***" libère la mémoire



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement
- Lorsqu'un objet n'est plus référencé : le "***Garbage collector***" libère la mémoire

Interpréteur

```
In  : a = 1 ↵
In  : b = 1 ↵
In  : b = "abc" ↵
In  : a = b ↵
In  : del a, b ↵
```

Espace des variables

a

b

Espace des objets

ref	2
type	str
id	0108
meth	...
attr	...

"abc"

Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement
- Lorsqu'un objet n'est plus référencé : le "***Garbage collector***" libère la mémoire

Interpréteur

```
In  : a = 1 ↵
In  : b = 1 ↵
In  : b = "abc" ↵
In  : a = b ↵
In  : del a, b ↵
```

Espace des variables

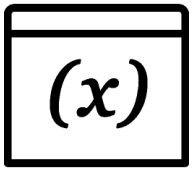
a

b

Espace des objets

ref	0
type	str
id	0108
meth	...
attr	...

"abc"



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement
- Lorsqu'un objet n'est plus référencé : le "***Garbage collector***" libère la mémoire

Interpréteur

```
In  : a = 1 ↵
In  : b = 1 ↵
In  : b = "abc" ↵
In  : a = b ↵
In  : del a, b ↵
```

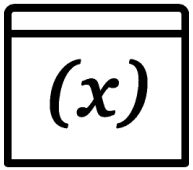
Espace des variables

a

b

Espace des objets

ref	0	"abc"
type	str	
id	0108	
meth	...	
attr	...	



Comment déclarer une variable sous Python ?

- Caractère "=" pour référencer un objet
- Type assigné dynamiquement
- Lorsqu'un objet n'est plus référencé : le "***Garbage collector***" libère la mémoire

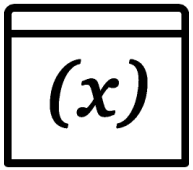
Interpréteur

```
In  : a = 1 ↵
In  : b = 1 ↵
In  : b = "abc" ↵
In  : a = b ↵
In  : del a, b ↵
```

Espace des variables

Espace des objets

ref	0	"abc"
type	str	
id	0108	
meth	...	
attr	...	



Comment déclarer une variable sous Python ?

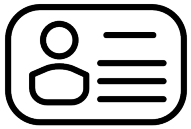
- Caractère "=" pour référencer un objet
- Type assigné dynamiquement
- Lorsqu'un objet n'est plus référencé : le "***Garbage collector***" libère la mémoire

Interpréteur

```
In  : a = 1 ↵
In  : b = 1 ↵
In  : b = "abc" ↵
In  : a = b ↵
In  : del a, b ↵
```

Espace des variables

Espace des objets



Quelles sont les règles de nommage ?

- Peut contenir les lettres majuscules et minuscules `[A-Z, a-z]` et les chiffres `[0-9]`
- Tous les autres caractères sont interdis à l'exception du caractère `"_"`
- Ne peut pas commencer par un chiffre
- Ne peut pas être l'un des mots suivants, réservés par Python :

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

Les mêmes règles s'appliquent aux noms de fichiers des scripts Python !!!

Note : *Choisissez des noms de variables explicites !*
Préférez un nom explicite comme `gamma` *plutôt que* `g`

Codage et décodage d'informations

Pour l'être humain, différents encodages pour un même objet :

13	*2 positions, 10 valeurs (10 nombres)*
treize	*6 positions, 26 valeurs (26 lettres)*
XIII	*4 positions, 7 valeurs ∈ [LVXICMD]*

En informatique, toute information est binaire :

- Par exemple, la séquence 1101001 1100110 peut représenter
  - la chaîne de caractères 'if' en ASCII
  - les entiers 105 et 102
  - les réels $0.13 \,.\, 10^{1}$ et $0.12 \,.\, 10^{6}$
  - ...

Nécessité de connaître le *type* de données pour le décodage !

Codage et décodage d'informations

Flux de bits 10100001111001111010011010001101111111011 10

Codage et décodage d'informations

Flux de bits	1010000111100111101001101000110111111101110					
Décodage 7 bits	1010000	1111001	1110100	1101000	1101111	1101110

Codage et décodage d'informations

Flux de bits	1010000111100111101001101000110111111101110					
Décodage 7 bits	1010000	1111001	1110100	1101000	1101111	1101110
Conversion décimale	80	121	116	104	111	110

Codage et décodage d'informations

Flux de bits	1010000111100111101001101000110111111101110					
Décodage 7 bits	1010000	1111001	1110100	1101000	1101111	1101110
Conversion décimale	80	121	116	104	111	110
Correspondance ASCII	P	y	t	h	o	n

Codage et décodage d'informations

Flux de bits	1010000111100111101001101000110111111101110					
Décodage 7 bits	1010000	1111001	1110100	1101000	1101111	1101110
Conversion décimale	80	121	116	104	111	110
Correspondance ASCII	P	y	t	h	o	n

Les caractères : tous les symboles existants sur les claviers du monde entier

De nombreuses tables de caractères existent :

- **ASCII standard** définit 2^7 (128) caractères communs à toutes les tables
- **ASCII étendu** en 8 bits fournit les caractères spéciaux (Français, Espagnol, ...)

Incompatibilité entre les différents types d'encodage étendus !

Erreurs connues lors de la lecture de mails, de fichiers, de pages web ...

Codage et décodage d'informations

Flux de bits	1010000	1111001	1110100	1101000	1101111	1101110
Décodage 7 bits	1010000	1111001	1110100	1101000	1101111	1101110
Conversion décimale	80	121	116	104	111	110
Correspondance ASCII	P	y	t	h	o	n

Les caractères : tous les symboles existants sur les claviers du monde entier

De nombreuses tables de caractères existent :

- **ASCII standard** définit 2^7 (128) caractères communs à toutes les tables
- **ASCII étendu** en 8 bits fournit les caractères spéciaux (Français, Espagnol, ...)

Incompatibilité entre les différents types d'encodage étendus !

Erreurs connues lors de la lecture de mails, de fichiers, de pages web ...

- **Unicode :** codage de l'intégralité des caractères existants (actuellement $\simeq 120\,000$)
  - ➤ UTF8 très utilisé, et compatible avec l'ASCII standard
  - ➤ UTF16/UTF32 utilisent des *mots* de 16/32 bits

Binaire	Déc	Écriture
010 0000	32	espace
010 0001	33	!
010 0010	34	"
010 0011	35	#
010 0100	36	$
010 0101	37	%
010 0110	38	&
010 0111	39	’
010 1000	40	(
010 1001	41	)
010 1010	42	*
010 1011	43	+
010 1100	44	,
010 1101	45	-
010 1110	46	.
010 1111	47	/
011 0000	48	0
011 0001	49	1
011 0010	50	2
011 0011	51	3
011 0100	52	4
011 0101	53	5
011 0110	54	6
011 0111	55	7
011 1000	56	8
011 1001	57	9
011 1010	58	:
011 1011	59	;
011 1100	60	<
011 1101	61	=
011 1110	62	>
011 1111	63	?

Binaire	Déc	Écriture
100 0000	64	@
100 0001	65	A
100 0010	66	B
100 0011	67	C
100 0100	68	D
100 0101	69	E
100 0110	70	F
100 0111	71	G
100 1000	72	H
100 1001	73	I
100 1010	74	J
100 1011	75	K
100 1100	76	L
100 1101	77	M
100 1110	78	N
100 1111	79	O
101 0000	80	P
101 0001	81	Q
101 0010	82	R
101 0011	83	S
101 0100	84	T
101 0101	85	U
101 0110	86	V
101 0111	87	W
101 1000	88	X
101 1001	89	Y
101 1010	90	Z
101 1011	91	[
101 1100	92	
101 1101	93	]
101 1110	94	^

Binaire	Déc	Écriture
101 1111	95	_
110 0000	96	‘
110 0001	97	a
110 0010	98	b
110 0011	99	c
110 0100	100	d
110 0101	101	e
110 0110	102	f
110 0111	103	g
110 1000	104	h
110 1001	105	i
110 1010	106	j
110 1011	107	k
110 1100	108	l
110 1101	109	m
110 1110	110	n
110 1111	111	o
111 0000	112	p
111 0001	113	q
111 0010	114	r
111 0011	115	s
111 0100	116	t
111 0101	117	u
111 0110	118	v
111 0111	119	w
111 1000	120	x
111 1001	121	y
111 1010	122	z
111 1011	123	{
111 1100	124	|
111 1101	125	}
111 1110	126	~

Codage et décodage d'informations

Les entiers : Exemple de l'ordinateur 3 bits

2^3 valeurs possibles	000	001	010	011	100	101	110	111	Maximum
Entier non signé	0	1	2	3	4	5	6	7	7
Entier signé	0	1	2	3	-4	-3	-2	-1	3

- Nombre de valeurs possibles fonction du nombre de bits :
  - nibble ou semioctet $2^4 = 16$
  - byte ou octet $2^8 = 256$
  - short $2^{16} = 65536$
  - long ou int $2^{32} = 4294967296$
  - long long ou int64 $2^{64} = 18446744073709551616$

- ***Note :*** Règle du *complément à 2* est une méthode pour signer les entiers
  - Échange des 0 et des 1 (2 = 010 devient 101)
  - Ajout de 1 en binaire (001 en 3 bits) (101 + 001 = 110 = -2)

Codage et décodage d'informations

Les réels : $1.637 = 1637 \times 10^{-3} = 0.1637 \times 10^{1} = 0.1637\text{e}1$

- Par convention, les nombres réels sont stockés sous la forme : $\pm 0.x\ e \pm y$
- Deux entiers signées x et y sont stockés :

		32 bits	*64 bits*
➤ Un pour l'exposant :	$\pm$ y	8 bits ($2^{\pm 127}$ max)	11 bits ($2^{\pm 1023}$ max)
➤ Un pour la mantisse :	$\pm$ x	23 bits ($\simeq$ 6 digits)	52 bits ($\simeq$ 15 digits)
		+ 1 bit de signe	

- Limitations :
  - ➤ Valeurs max. et min. imposées par l'ordinateur
  - ➤ Précision limitée (nombre imposé de chiffres significatifs)
  - ➤ Certains nombres ne peuvent être stockés de manière exacte ($1/3$, π, $\sqrt{2}$, ...)

Les types numériques sous Python

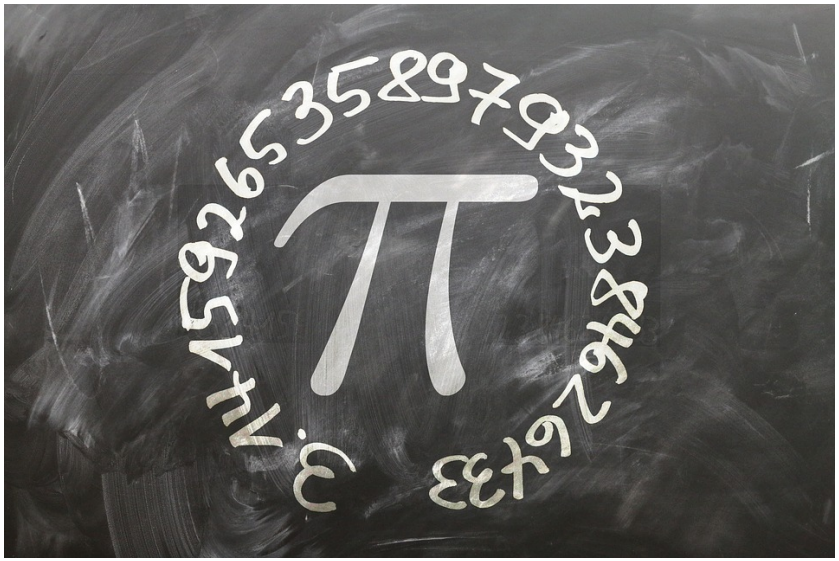


Image from Pixabay (Geralt)

Il existe 4 types numériques de base sous Python :

- **Le type `int` utilisé pour représenter les nombres entiers**
  - Précision arbitraire : uniquement limitée par la mémoire disponible
  - constructeur éponyme `int`()
- **Le type `float` utilisé pour représenter les nombres réels**
  - précision dépendante du système
  - constructeur éponyme `float`()
- **Le type `complex` utilisé pour représenter les nombres complexes**
  - composé de deux `float` : partie réelle/partie imaginaire
  - constructeur éponyme `complex`() : `complex`([real[, imag]])
- **Le type `bool` utilisé pour représenter les 2 booléens `True` et `False`**
  - sous type de `int`
  - `False` correspond à l'entier 0
  - `True` correspond à l'entier 1
  - constructeur éponyme `bool`()

Important : Les objets de ces différents types numériques sont des objets ***immuables***
Les méthodes qui les modifient en modifient une copie à la place

Opérations supportées par les objets de types `complex`, `float` et `integers`			
	Opération	**Opérateur**	**Description**
	Addition	`a + b`	Somme de **a** et **b**
	Soustraction	`a - b`	Soustraction de **a** et **b**
	Multiplication	`a * b`	Multiplication de **a** par **b**
	Division	`a / b`	Division de **a** par **b**
	Puissance	`a**b` ou `pow(a, b)`	**a** à la puissance b
	Module	`abs(a)`	Module de **a**
*	Modulo	`a%b`	Reste de la division de **a** par **b**
*	Division entière	`a//b`	Division entière de **a** par **b**
*	Division “complète”	`divmod(a, b)`	Retourne le tuple `(a//b, a%b)`
	Conjugué	`a.conjugate()`	Méthode retournant le conjugué de **a**
*	Conversion entière	`int(a)`	Troncation entière de **a**
*	Conversion réelle	`float(a)`	Convertit **a** en `float`
	Conversion complexe	`complex(a)`	Convertit **a** en `complex`

* Ces opérations ne sont pas supportées par les objets de type `complex`

```
# Déclaration avec le signe '='
In  : a = 2                 # Crée l'entier 2, a réfère cet objet
In  : b = 2.0               # Crée le réel 2.0, b réfère cet objet
In  : c = 2+2j              # Crée le complexe 2+2j, c réfère cet objet

# Typage dynamique :
In  : type(a + b)           # Opération impliquant plusieurs types numériques
Out : float                 # donne le type le plus "fort" au résultat
In  : type(a + b + c)
Out : complex

# Précision :
In  : d = a**512            # Objets int de précision illimitée !
In  : print(d)
Out : 179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624224137216
In  : b*d                   # La conversion peut conduire ...
Out : 2.6815615859885194e+154
In  : c*d                   # ...à une perte de précision
Out : (2.6815615859885194e+154+2.6815615859885194e+154j)
```

Illustration : notion d'héritage

```
# Cas des float
In  : a = 1.5
In  : a.as_integer_ratio()   # La méthode as_integer_ratio()...
Out : (3, 2)                 # ...parle d'elle même

# Cas des complex
In  : b = 1+2j               # Crée le complexe 1 + 2j, b réfères cet objet
In  : c = complex(1, 2)      # Écriture alternative
In  : c.conjugate()          # Méthode conjugate() héritée ...
Out : (1-2j)                 # ...de la classe complex
In  : c.real                 # Attribut real hérité de la classe complex
Out : 1
In  : c.imag                 # Attribut imag hérité de la classe complex
Out : 2
```

- On utilise le point (.) sous Python pour accéder à l'héritage d'un objet
- Un **attribut** est une caractéristique de l'objet : son appel se fait **sans parenthèses**
- Une **méthode** est une fonction qui s'applique à l'objet : son appel se fait **avec des parenthèses et d'éventuels arguments**

Les structures de données

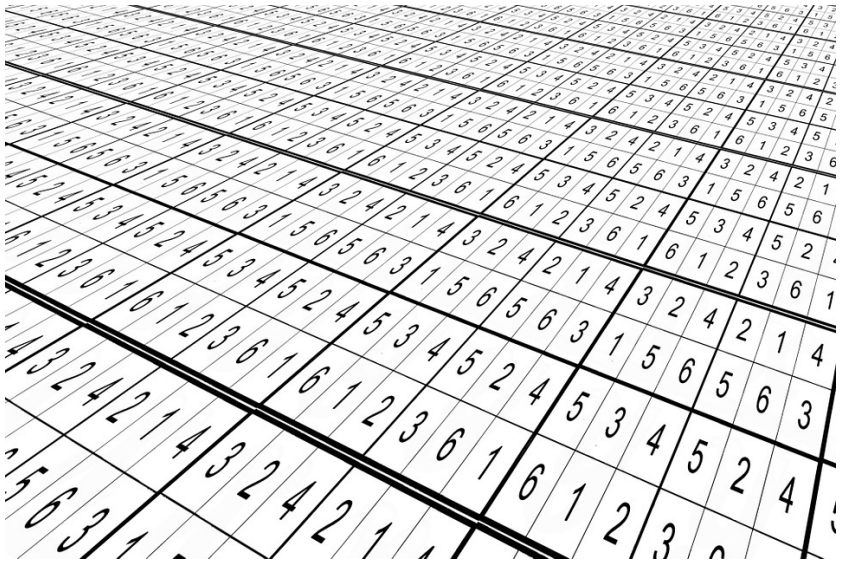


Image from Pixabay (Geralt)

Liste des types `séquentiels`

Les types séquentiels natifs sont :

- **Le type `string`** : séquence *immuable* de caractères
  - type particulier de séquence appelé chaîne de caractères
  - déclaration avec `' '` ou `" "` ou `''' '''`
  - conversion avec le constructeur `str()`
- **Le type `list`** : ensemble ordonnée *muable* d'éléments
  - déclaration avec `[]`
  - conversion avec le constructeur `list()`
- **Le type `tuple`** : ensemble ordonné *immuable* d'éléments
  - déclaration avec `()`
  - conversion avec le constructeur `tuple()`
- **Le type `range`** : séquence *immuable* de nombres
  - déclaration avec le constructeur `range()`
  - couramment utilisée dans les boucles `for` en tant qu'itérateur
- **Les séquences binaires** utilisées pour manipuler les données binaires
  - le type `bytes` : séquence *immuable* d'octets
  - le type `bytearray` : équivalent *muable* des objet `bytes`
  - le type `memoryview` : utilisé pour accéder aux données internes d'un objet

Les '''chaînes de caractères'''

Illustration : Déclaration et opérations

```
# Déclaration :
In  : s1 = "Hello World "
In  : s2 = "7"
In  : type(s1)                  # isinstance(s1, str) == True
Out : str
In  : type(s2)                  # isinstance(s2, str) == True
Out : str
In  : print(s1, s2)
Out : Hello World  7

# Opérations supportées :
In  : s1 + s2                   # Concaténation
Out : "Hello World 7"
In  : s1*3                      # Répétition
Out : "Hello World Hello World Hello World"

# Objet immuable : ne peut pas changer !
In  : s1[1] = "a"
Out : TypeError:  'str' object does not support item assignment
```

Les '''chaînes de caractères'''

Illustration : Héritage

```
# Les objets de type str héritent de très nombreuses méthodes :
In  : s1.split()                    # split() retourne une liste
Out : ["Hello", "World"]

In  : s1.replace("World", "Bro")    # La méthode replace()
Out : "Hello Bro "

In  : s1.center(30)                 # La méthode center()
Out : "          Hello World          "

In  : s1.rjust(30)                  # La méhode rjust()
Out : "                   Hello World "

In  : s1.upper()                    # La méthode upper()
Out : "HELLO WORLD "

In  : s1.lower()                    # La méthode lower()
Out : "hello world "
                                    # ... et bien d'autres...
```

Les '''chaînes de caractères'''

Concaténer des objets de types différents

```
In  : apples = 22
In  : mystr = "There are " + str(apples) + " apples"
In  : print(mystr)
Out : There are 22 apples
```

Formatage

```
In  : from math import pi

# La méthode format :
In  : mystr = "Pi equals {:.2f}, more precisely {:.5f}"
In  : print(mystr.format(pi, pi))
Out : Pi equals 3.14, more precisely 3.14159

# Les f-strings (Python >= 3.6) :
In  : mystr = f"Pi equals {pi:.2f}, more precisely {pi:.5f}"
In  : print(mystr)
Out : Pi equals 3.14, more precisely 3.14159
```

Les [listes]

Illustration : Déclaration et manipulations

```
# Déclaration :
In  : bestbeers = ["Guinness", "Chouffe", "86"]
In  : type(bestbeers)                  # isinstance(bestbeers, list) == True
Out : list
In  : print(bestbeers)
Out : ['Guinness', 'Chouffe', '86']

# Objet muable : ajout, suppression, modification possible
In  : bestbeers.pop(2)                 # Supprime la valeur n^o 2
In  : print(bestbeers)
Out : ["Guinness", "Chouffe"]

In  : bestbeers.append("DeuS")         # Ajoute une nouvelle valeur à la fin
In  : print(bestbeers)
Out : ["Guinness", "Chouffe", "DeuS"]

In  : bestbeers[1] = "Kro"             # Modifie la valeur d'indice 1
In  : print(bestbeers)
Out : ["Guinness", "Kro", "DeuS"]
```

Les [listes]

Illustration : Héritage et opérations

```
# Les objets de type list héritent de nombreuses méthodes :
In  : bestbeers.sort()                  # Tri la liste in-place
Out : ['DeuS', 'Guinness', 'Kro']
In  : bestbeers.insert(1, "Leffe")      # Nouvel élément à l'indice 1
Out : ['DeuS', 'Leffe', 'Guinness', 'Kro']

# Chaque élément d'une liste peut être d'un type différent...
In  : mix1 = [2.1, "abc"]
In  : mix2 = [ [10, "cba"], 1]

# ... et hérite donc des méthodes de sa classe
In  : mix1[1].upper()
Out : 'ABC'

# Répétition et concaténation avec + et * comme pour les objets str
In  : mix1 + mix2
Out : [2.1, "abc", [10, "cba"], 1]
In  : mix1*2
Out : [2.1, "abc", 2.1, "abc"]
```

Les (tuples,)

Illustration

```
# Déclaration :
In  : days = ("monday", "tuesday", "wednesday")
In  : type(days)            # isinstance(days, tuple) == True
Out : tuple
In  : print(days)
Out : ('monday', 'tuesday', 'wednesday')

# Objet immuable :  ne peut pas changer !
In  : days[1] = "sunday"
Out : TypeError:  'tuple' object does not support item assignment

# En fait, les parenthèses sont optionnelles. La virgule définit le tuple !
In  : days = "monday", "tuesday", "wednesday"
In  : type(days)
Out : tuple

# Chaque élément d'un tuple peut être d'un type différent
In  : mix = 2.1, 4+3j, "abc", [10, "cba"], ("10", "100"), 1
```

Le type range

Syntaxe du constructeur range:

- `range(n)` génère une séquence de n `int`, commençant à 0
- `range(n, m)` génère une séquence de $m - n$ `int`, de n à $m - 1$
- `range(n, m, s)` génère une séquence d'`int` de n à $m - s$ par pas de s
- les arguments n et s sont optionnels ($n = 0$ et $s = 1$ par défaut)
- avantage : faible empreinte mémoire (seuls n, m, et s sont stockés)

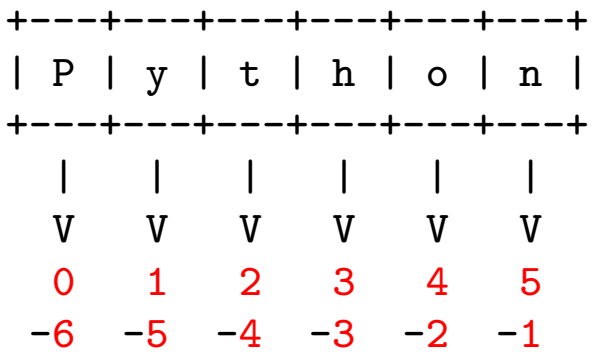
Illustration

```
# Déclaration :
In  : r = range(5)          # permet de générer la séquence 0, 1, 2, 3, 4
In  : type(r)               # isinstance(r, range) == True
Out : range
In  : print(r)              # valeurs générées à la demande uniquement
Out : range(0, 5)

In  : range(1, 10, 2)       # permet de générer la séquence 1, 3, 5, 7, 9
Out : range(1, 10, 2)
In  : range(10, 1, -2)      # permet de générer la séquence 10, 8, 6, 4, 2
Out : range(10, 1, -2)
```

L'Indexation sous Python

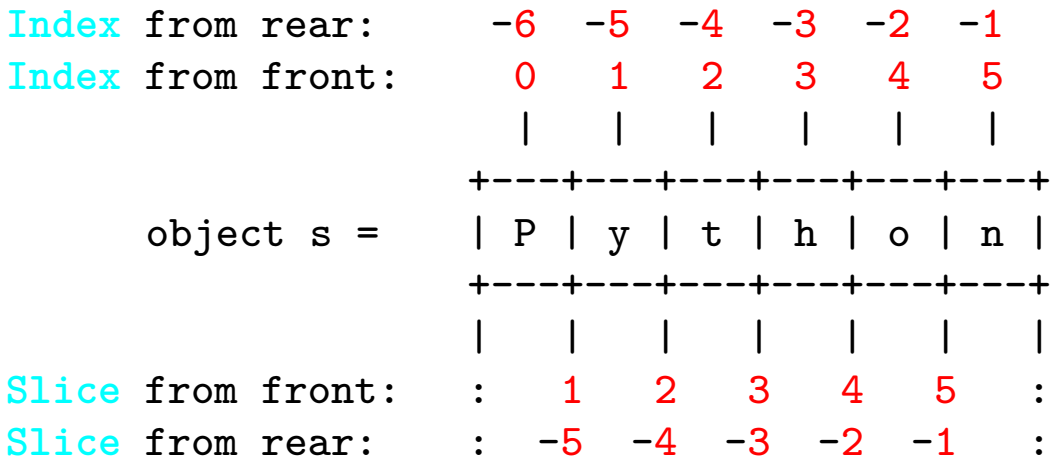
- Comme dans la plupart des langages, Python utilise `[ ]` pour l'indexation
- Comme dans la plupart des langages, Python utilise une indexation basée sur 0
- Python utilise également des indices négatifs : le dernier élément d'un conteneur itérable est l'élément d'indice -1




```
In  : a = [3, 5, 7, 9]
In  : a[0]
Out : 3
In  : a[-1]
Out : 9
In  : a[-2]
Out : 7
```

```
In  : a = "Hello World!"
In  : a[0]
Out : H
In  : a[-1]
Out : !
In  : a[-2]
Out : d
```

Le `slicing` : un moyen rapide et méthodique pour accéder aux données



Syntaxe : `sliceable_object[<start>:<end>:step]`

```
» s[0]    # out:  "P"
» s[5]    # out:  "n"
» s[-1]   # out:  "n"
» s[-2]   # out:  "o"
```

```
» s[1:]      # out:  "ython"
» s[:5]      # out:  "Pytho"
» s[:-2]     # out:  "Pyth"
» s[-5:-2]   # out:  "yth"
```

- `<end>` représente le dernier élément qui **ne fait pas partie** du slice
- `<start>`, `<end>`, et `step` peuvent être négatifs et sont **tous optionnels**

Le slicing : un moyen rapide et méthodique pour accéder aux données

Écriture littérale

```
In  : [1, 2, 3, 4, 5, 6][2:]   # Pas de restriction sur l'utilisation...
Out : [3, 4, 5, 6]             # de 'literals'...
```

Quelques exemples ...

```
» s = [0, 1, 2, 3, 4]
» s[::-1]    # [4, 3, 2, 1, 0]
» s[::-2]    # [4, 2, 0]
» s[1::-2]   # [1]
» s[2::-2]   # [2, 0]
```

...plus complexes

```
» s = "Hello World!"
» s[::-1]    # "!dlroW olleH"
» s[::-2]    # "!lo le"
» s[1::-2]   # "e"
» s[2::-2]   # "lH"
```

Définition externe d'un slice : slice(start, stop[, step])

```
In  : myslice = slice(None, None, -2)   # None pour valeur par défaut
In  : s[myslice]                        # Équivalent à s[::-2]
Out : [4, 2, 0]
```

Note: Quand step est négatif, la valeur par défaut pour <start> est len(s) (tandis que <end> n'est pas égal à 0 car s[...-1] contient s[0]). Que donne a[3::-3] ?

Les types `d'ensemble` et de `correspondance`

Les types de correspondance : *indexation par clés*

- **Le type `dict`** : collection *muable* non ordonnée d'éléments
  - déclaration avec `{}`
  - conversion avec le constructeur `dict()`

Les types d'ensemble : *pas d'indexation*

- **Le type `set`** : collection *muable* non ordonnée d'éléments *uniques*
  - déclaration avec `{}`
  - `set` vide déclaré avec `set()` : évite la confusion avec dictionnaire vide `{}`
  - conversion avec le constructeur `set()`
- **Le type `frozenset`** : collection *immuable* non ordonnée d'éléments *uniques*
  - déclaration avec `frozenset()`
  - conversion avec le constructeur `frozenset()`
- Généralement utilisés pour le calcul d'opérations mathématiques sur des ensembles

Les {'dictionnaires':'mapping object'}

Illustration

```
# Déclaration :  clés = données de type hashable (≃ immuable)
In  : d = {"int":1, "float":2.0, "list":["a", 3], 0:0}
In  : type(d)      # isinstance(d, dict) == True
Out : dict
In  : print(d)     # Ensemble d'éléments non ordonnés
Out : {'list':  ['a', 3], 'int':  1, 'float':  2.0, 0:  0}

# Objet muable :  possibilité d'ajouter, enlever, modifier des éléments
In  : d["dict"] = {(0, 1):  "tup1", (0, 2):  "tup2"}
In  : print(d)
Out : {0:  0, 'dict':  {(0, 1):  'tup1', (0, 2):  'tup2'}
       'float':  2.0, 'int':  1, 'list':  ['a', 3]}

# Les objets de type dict héritent de nombreuses méthodes :
In  : d.keys()     # Liste les clés du dictionnaire
Out : dict_keys(['list', 'int', 'float', 'dict', 0])
In  : d.values()   # Liste les valeurs du dictionnaire
Out : dict_values([{(0, 1): 'tup1', (0, 2): 'tup2'}, 2.0, 1, ['a', 3]], 0)
```

Les {sets}

Illustration

```
# Déclaration :
In  : s1 = {2, 1, 1, 3}      # Équivalent à :  set([2, 1, 1, 3])
In  : type(s1)
Out : set                    # isinstance(s1, set) == True
In  : print(s1)
Out : {1, 2, 3}              # Ensemble d'éléments uniques et non ordonnés

# Les objets de type set héritent de nombreuses méthodes :
In  : s2 = {3, 5, 1, 8}
In  : s1.intersection(s2)
Out : {1, 3}
In  : s1.difference(s2)
Out : {2}
In  : s1.symmetric_difference(s2)
Out : {2, 5, 8}
In  : s1.union(s2)
Out : {1, 2, 3, 5, 8}
In  : s1.add(4)              # Objet muable :  possibilité...
Out : {1, 2, 3, 4}           # ...d'ajouter ou d'enlever des éléments
```

Opérateur	Description
	Opérations communes sur les séquences
`x in s`	`True` si `x` est dans `s`, `False` sinon
`x not in s`	`False` si `x` est dans `s`, `True` sinon
`s + t`	Concaténation de `s` et `t`
`s * n`	Répète `s` `n` fois
`s[i]`	i-ème élément de `s` (en commençant à 0)
`s[i:j:k]`	Tranche de `s` de `i` à `j` par pas de `k`
`len(s)`	Longueur de `s`
`min(s)`	Plus petit élément de `s`
`max(s)`	Plus grand élément de `s`
`s.count(x)`	Nombre d'occurences de `x` dans `s`
`s.index(x[, i[, j]])`	Indice de la première occurence de `x` dans `s` (`i` et `j` : bornes optionnelles)
	Opérations sur les séquences muables
`s[i] = x`	Élément `i` de `s` est remplacé par `x`
`s[i:j] = t`	Tranche de `s` de `i` à `j` est remplacée par le contenu de l'itérable `t`
`del s[i:j]`	Identique à `s[i:j] = []`
`s[i:j:k] = t`	Éléments de `s[i:j:k]` sont remplacés par ceux de t
`del s[i:j:k]`	Supprime les éléments de `s[i:j:k]` de la liste
`s.append(x)`	Ajoute `x` à la fin de la séquence
`s.clear()`	Supprime tous les éléments de `s` (identique à idel `s[:]`)
`s.copy()`	Crée une copie superficielle de `s` (identique à `s[:]`)
`s.extend(t)`	Étend `s` avec le contenu de `t` (identique à `s += t`)
`s *= n`	Met à jour `s` avec son contenu répété `n` fois
`s.insert(i, x)`	Insère `x` dans `s` à l'index donné par `i`
`s.pop([i])`	Récupère l'élément à `i` et le supprime de `s`
`s.remove(x)`	Supprime le premier élément de `s` pour qui `s[i] == x`
`s.reverse()`	Inverse sur place les éléments de `s`

Extrait de https://docs.python.org

Unpack : Répartir des élément d'un itérable dans plusieurs variables

```
» letters = ("a", "b", "c")
» letter1 = letters[0]
» letter2 = letters[1]
» letter3 = letters[2]
```

Unpacking : nb variables == nb d'éléments de l'itérable

```
» letter1, letter2, letter3 = letters
» print(letter1)                          # Out :  "a"
» print(letter2)                          # Out :  "b"
» print(letter3)                          # Out :  "c"
```

Unpacking : nb variables != nb d'éléments de l'itérable

```
» letter1, *others = letters
» print(letter1)                  # Out :  "a"
» print(others)                   # Out :  ["b", "c"] - (C'est une liste !)
```

- Multiplication `*` et puissance `**` pour les types numériques
- Répétition `*` pour les structures
- Opérateur *Splat* `*` pour *unpacker* des itérables
- Opérateur *Double Splat* `**` pour le cas spécial de l'*unpacking* des `dict`
  (si un simple *splat* est utilisé, l'*unpacking* agira sur les clés et non pas les valeurs !)

Les autres types natifs principaux

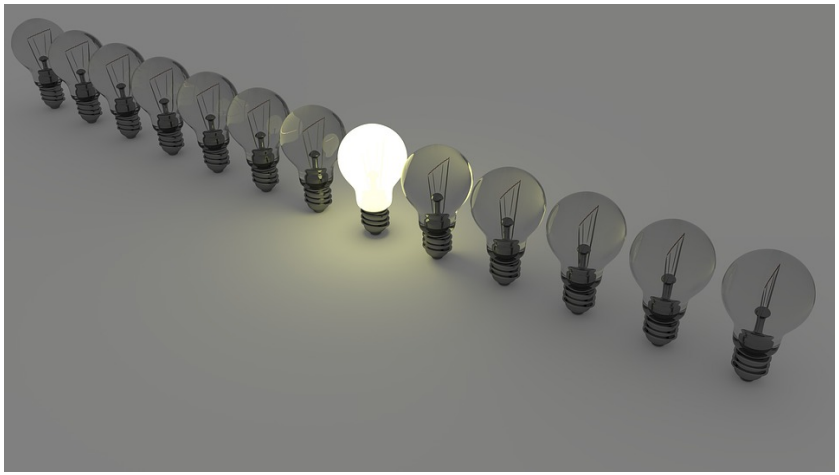


Image from Pixabay (Stevepb)

Les fonctions *(vues plus tard)*

```
In  : def f(x):
          return x**2
In  : type(f)
Out : function
```

Les classes et instances de classe *(vues plus tard)*

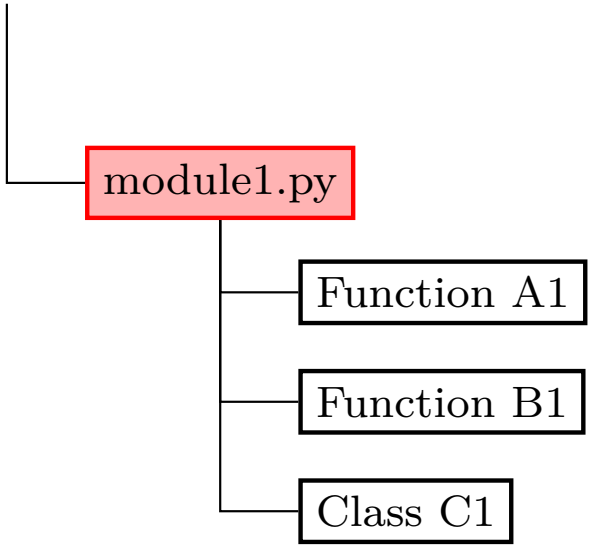
```
In  : class MyClass:
          pass
In  : type(MyClass)
Out : type
In  : instance = MyClass()
In  : type(instance)
In  : __main__.MyClass
```

Les modules

```
In  : import numpy
In  : type(numpy)
Out : module
```

Qu'est ce qu'un `Module` ?

- Un fichier `*.py` contenant des définitions et instructions (fonctions, classes, ...)
- Le nom du fichier est celui du module avec le suffixe `.py`
- Python fournit de base une bibliothèque de modules standards

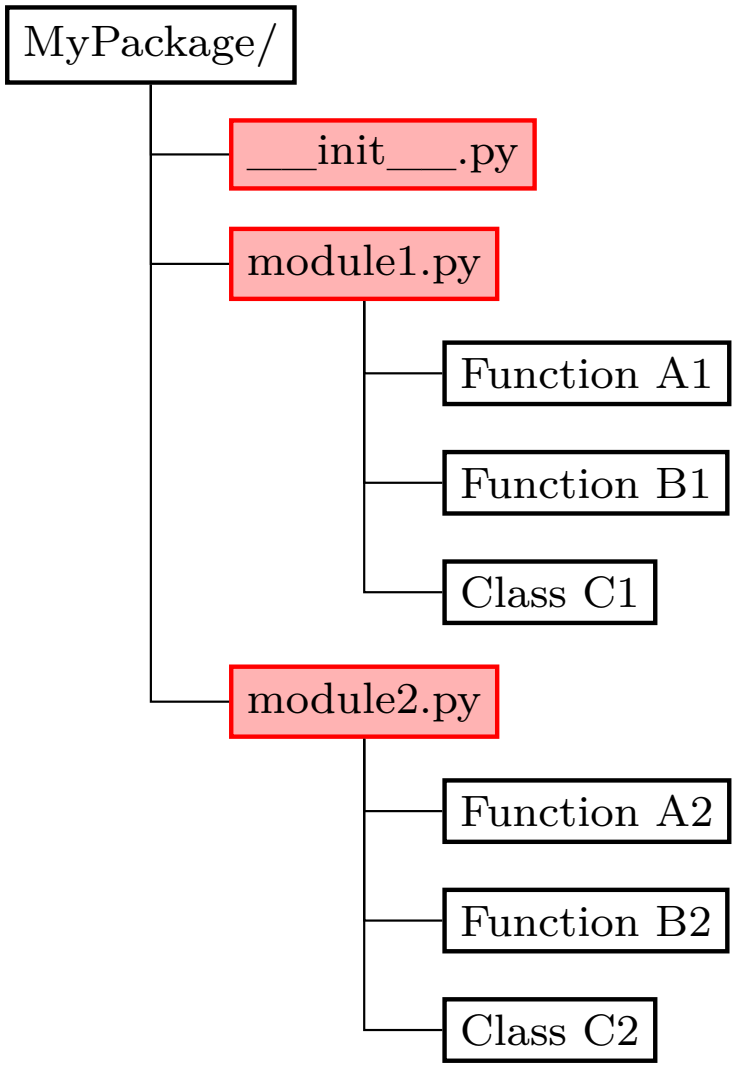


Qu'est ce qu'un `Module` ?

- Un fichier `*.py` contenant des définitions et instructions (fonctions, classes, ...)
- Le nom du fichier est celui du module avec le suffixe `.py`
- Python fournit de base une bibliothèque de modules standards

Qu'appelle-t-on un `Package` ?

- Une collection de modules organisée hiérarchiquement
- Un package contient toujours un fichier `__init__.py`
- `__init__.py` assure la distinction entre un package et un simple répertoire rempli de scripts
- Les packages peuvent être imbriqués jusqu'à n'importe quelle profondeur
- Chaque package imbriqué doit contenir son propre fichier `__init__.py`



Python vient avec de nombreux modules appelés modules standards :

- Interactions avec l'os ou le système : `sys`, `os`, `time`, ...
- Protocoles internet : `urllib`, `sockerserver`, ...
- Exécution simultanée : `threading`, `multiprocessing`, ...
- Mathématiques : `math`, `statistics`, ...
- ...

L'immense communauté développe activement de nombreux autres modules :

- Jeux vidéos : `pygame`, `pyglet`, ...
- Multimédia : `PIL`, `cv2`, ...
- General User Interface (GUI) : `wxPython`, `PyGtk`, ...
- Optimisation : `Cython`, `numba`, ...
- ...

Les scientifiques ne sont pas en reste :

- Calcul numérique : `numpy`
- Calcul scientifique : `scipy`
- Calcul symbolique : `sympy`
- Librairie graphique : `matplotlib`
- ...

Comment utiliser ces modules ?

- En les important dans l'espace des variables !
- Le mot clé `import` permet de charger un module
- Les mots clés `from`, `as`, le *splat operator* `*` permettent une importation plus pratique

Illustration

```
In  : import math                 # Importe le module math
In  : math.cos(math.pi)           # Retourne -1.0

In  : import math as mt           # math est maintenant attaché au nom...
In  : mt.cos(mt.pi)               # ...mt dans l'espace local des variables

In  : from math import *          # Importe toutes les classes et méthodes
In  : cos(pi)                     # cos vient du module math

In  : from math import cos, pi    # Importe seulement la méthode cos...
In  : cos(pi)                     # ...et l'attribut pi

In  : import sys, math            # Importe les modules sys et math
```

Quelques mots sur les bonnes pratiques ...

- Elle dépendent généralement de ce dont vous avez besoin dans le module !
- Besoin d'une seule fonction : vous pouvez importer uniquement cette fonction
- De manière générale, gardez un lien vers les modules utilisés dans l'espace local !

Évitez les imports globaux comme : `from math import *`

- Surcharge l'espace des variables et restreint les noms de variables disponibles
- Provoque des conflits si des objets homonymes existent dans des modules différents

```
In  : from numpy import *                          # Importe tout du module numpy
In  : cos.__class__
Out : <type 'numpy.ufunc'>

In  : from math import *                           # Importe tout du module math
In  : cos.__class__                                # la méthode cos de numpy...
Out : <type 'builtin_function_or_method'>          # ...a été surchargée
```

Le type ndarray et le calcul scientifique

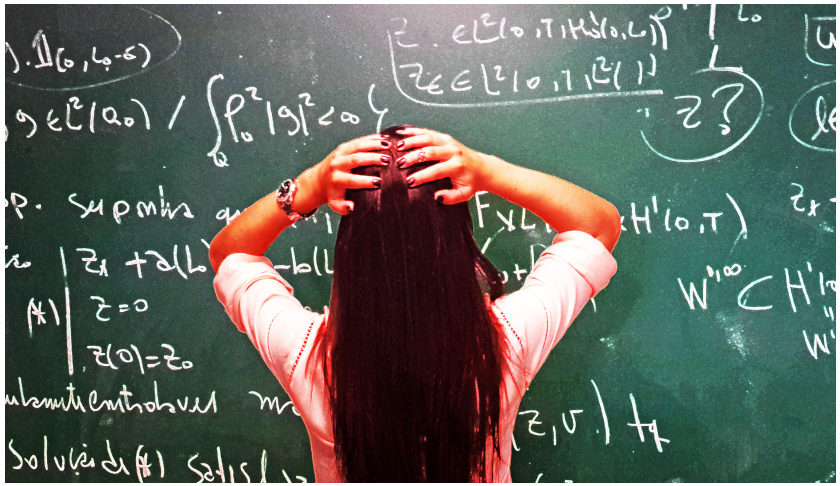


Image under Creative Common licence

Opérations vectorielles : indirectes avec les types de base (`list`, `tuple`, `dict`, `set`, ...) :

```
In  : [1, 2] * 2          # Répétition
Out : [1, 2, 1, 2]
In  : [1, 2] + [3, 4]     # Concaténation
Out : [1, 2, 3, 4]
In  : [1, 2] / 2
Out : TypeError:  unsupported operand type(s) for /:  'list' and 'int
In  : [1, 2] ** 2
Out : TypeError:  unsupported operand type(s) for **:  'list' and 'int
```

Opérations vectorielles : indirectes avec les types de base (`list`, `tuple`, `dict`, `set`, ...) :

```
In  : [1, 2] * 2          # Répétition
Out : [1, 2, 1, 2]
In  : [1, 2] + [3, 4]     # Concaténation
Out : [1, 2, 3, 4]
In  : [1, 2] / 2
Out : TypeError:  unsupported operand type(s) for /:  'list' and 'int
In  : [1, 2] ** 2
Out : TypeError:  unsupported operand type(s) for **:  'list' and 'int
```

Module spécifique Numpy : dédié au calcul numérique

- Fournit le type `ndarray` (*n*-dimensional array)
- Fournit le support de calculs vectoriels et matriciels (*math.* ou *'element-wise'*)
- Fournit la méthode `array` classiquement utilisée pour déclarer des *matrices*

Illustration

```
In  : import numpy as np
In  : myarray1D = np.array([1, 3, 2, 7])
In  : myarray1D**2                          # Puissance élément à élément
Out : array([1 9 4 49])
```

Illustration : Méthodes et attributs

```
# Création d'une array de dimension (2, 4) :
In  : arr = np.array([[1, 3, 2, 7], [11, 13, 12, 17]])

# Les objets ndarray héritent de nombreuses méthodes et attributs
In  : arr.shape              # Forme de l'objet ...
Out : (2, 4)
In  : arr.max()              # Valeur maximale dans arr ...
Out : 17
In  : arr.argmax()           # ...Indice de cette valeur max.  ...
Out : 7
In  : arr.mean()             # ...Moyenne de tous les éléments ...
Out : 8.25
In  : arr.std()              # ...et écart-type ...
Out : 5.4943152439589777
```

Les objets de type `ndarray` héritent de nombreuses méthodes :

- Outils de conversion et de manipulation de forme
- Sélection et manipulation d'éléments
- Arithmetique, multiplication matricielle, opérations de comparaison
- ...Cf. C3 pour plus de détails : *Le calcul scientifique*

Résumé des types d'objets sous Python

Main built-in types in Python

Cat.	Type	Convert	Create	Mut.	Index
Num.	int	int()	1	no	num
Num.	float	float()	1.0	no	num
Num.	complex	complex()	1+2j	no	num
Seq.	list	list()	[1, "a"]	yes	num
Seq.	tuple	tuple()	(1, "a") or 1, "a"	no	num
Seq.	range	✗	range(bg[,end][,stp])	yes	num
Set	set	set()	{0, 1}	yes	✗
Set	frozenset	frozenset()	frozenset()	no	✗
Txt seq.	str	str()	' ', " " or """ """	no	num
Map	dict	dict()	{"k1":1, "k2":"a" }	yes	key
Other	bool	bool()	True/False	no	✗
Other	module	✗	import	yes	✗
Other	class	✗	class	yes	✗
Other	method	✗	def	yes	✗
+ np	ndarray	array()	array(list)	yes	num

Note : Sans oublier les types séquences binaires : bytes, bytearray, memoryview


TO BE
CONTINUED...