Algorithmique et Programmation

Introduction ; Types et variables

Elise Bonzon

elise.bonzon@mi.parisdescartes.fr

LIPADE - Université Paris Descartes
http://www.math-info.univ-paris5.fr/∽bonzon/

Types et variables

1. Organisation du cours
2. Différents types de langages
3. Le langage Python
4. Variables
5. Les types de données
6. Entrées/Sorties
7. Pour conclure

Organisation du cours

Organisation du cours

- Cours : 1h30 par semaine. Amphi Binet, Lundi 10h45-12h15
- 1h30 de TD et 1h30 de TP par semaine

Les horaires et les salles de cours sont susceptibles de changer au cours du semestre, vérifiez régulièrement votre emploi du temps sur l'ENT.

- Les transparents de cours et les sujets de TD et TP sont disponibles sur Moodle
- Les informations importantes sur le cours seront également affichées à cet endroit : allez vérifier fréquemment !

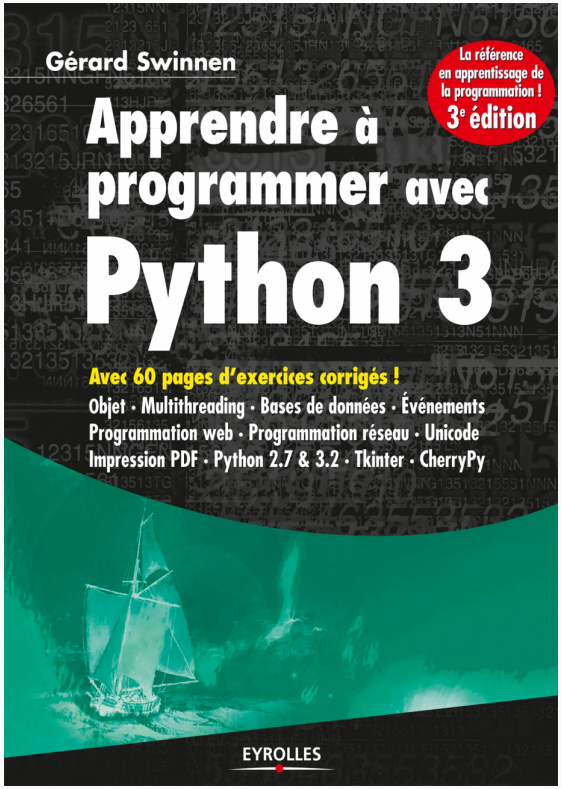
Contrôle des connaissances

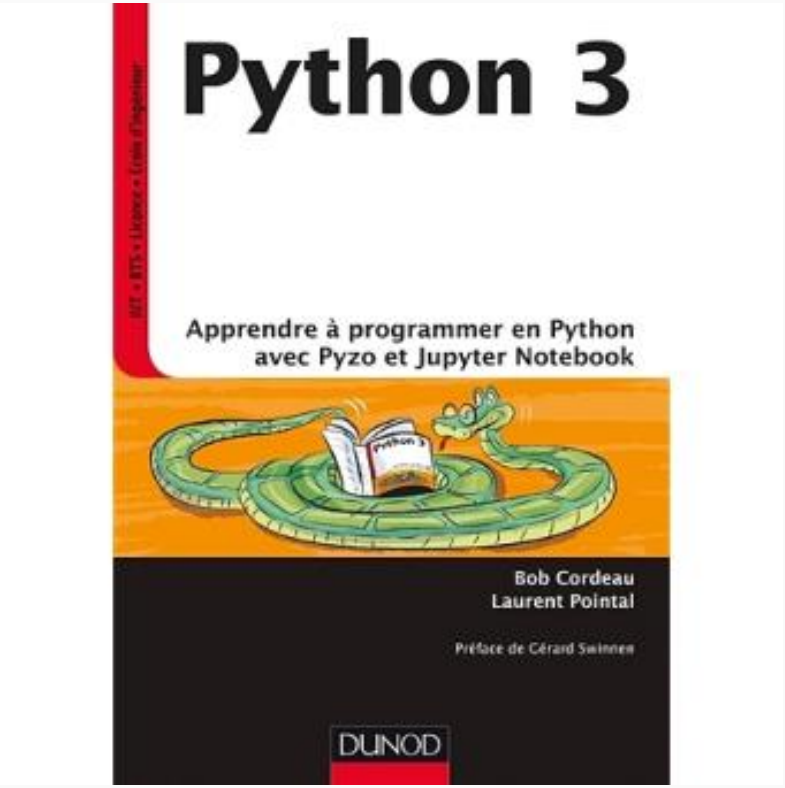
- Contrôle continu CC
- Attention : les dates sont **sous réserve de modification** :
  - 3 QCMs dans le semestre, en début des TDs des semaines 5, 8 et 12.
    La moyenne de ces 3 notes est appelée CC_{QCM}
  - CC_M : Un examen machine en semaine **13**
  - CC_F : Une épreuve écrite de 1h30 en janvier
- Note session 1 $= CC_{QCM} * 0.3 + CC_M * 0.3 + CC_F * 0.4$
- Note session 2 $= CC_F$

Conseils pour réussir ce cours

- **Ne pas se sous-estimer** : pas de pré-requis informatiques sont nécessaires pour ce cours
- **Ne pas se sur-estimer** : des connaissances en informatique et en Python ne garantissent pas de réussir
- **Aller en cours**
- **Etre actif** en TD et en TP
- **Travailler seul** : relire les cours le soir, refaire les exercices de TD et TP, et faire ceux non traités en TD et en TP

Bibliographie partielle





G. Swinnen. *Apprendre à programmer avec Python 3*. Editions. Eyrolles.

B. Cordeau et L. Pointal. *Python 3 - Apprendre à programmer en Python avec PyZo et Jupyter Notebook*. Editions Dunod.

De nombreuses ressources existent en Python, que ce soit des livres ou sur internet. N'hésitez pas à aller voir !

Différents types de langages

Programmation

- **Langage machine** (appelé aussi *code machine*, *code natif* ou *code binaire*) : directement compréhensible par la machine. Codage purement numérique représentant des opérations très basiques et/ou des données.
- **Langage d'assemblage** : langage symbolique textuel d'un peu plus haut niveau, traduisible en langage machine par un programme **assembleur**.
- **Langage de haut niveau** : doit être compilé ou interprété pour être compris par la machine

Langages de programmation

Langage de programmation

Un langage de programmation est un langage informatique, permettant à un humain d'écrire un code source qui sera analysé par un ordinateur.

Paradigmes de programmation

Il existe plusieurs paradigmes de programmation, c'est à dire des logiques de fonctionnement :

- **Programmation impérative** : l'exécution d'une séquence d'instructions va modifier l'état des données (la plupart des langages sont impératifs)
- **Programmation objet** : à base de classes, façon d'organiser et de regrouper les données et les traitements qui y sont liés
- **Programmation fonctionnelle** : déclarative, qui se rapproche des théories mathématiques

Modes d'execution des langages de programmation

Il existe trois modes d'execution des langages de programmation :

- les langages **compilés**
- les langages **interpretés**
- les langages **semi-compilés** (ou *à bytecode*)

Les langages compilés

Compilation

- Exécutée par un **programme**, appelé compilateur
- Traduction du code source en une fois en code machine
- Le code machine est directement exécutable par la machine, indépendamment du compilateur
- Peut être conservé tel quel dans un fichier (*fichier exécutable*).

Par exemple, les langages Pascal, Ada, C, C++, Fortran...

Les langages interprétés

Interprétation

- Exécutée par un **programme**, appelé interpréteur
- Chaque ligne du code source est traduite au fur et à mesure en instructions directement exécutées
- Pas de génération de programme objet
- Technique très souple, mais peu performante : l'interprétation doit être réalisée à chaque exécution

Par exemple, les langages LISP, Basic, Perl, PROLOG...

Les langages semi-compilés

Semi-compilation

- Combinaison des deux techniques précédentes
- Le compilateur produit un code intermédiaire appelé *bytecode*
- L'interpréteur l'exécute pour produire le résultat
- Le bytecode est facile à interpréter en langage machine

Par exemple, les langages Java, Python...

Le langage Python

Le langage Python

Python

Le langage Python a été créé en 1989 par Guido van Rossum. C'est un langage **semi-compilé**, **orienté objet**.

Il en existe plusieurs versions. Nous travaillerons dans ce cours en Python 3.

Python, un langage utilisé ?

Le langage Python est utilisé dans de nombreuses applications. Par exemple :

- BitTorrent, client original, avec de nombreux dérivés
- Dropbox, un service web d'hébergement de fichiers
- Juice, un téléchargeur de podcasts populaire
- OpenShot Video Editor
- Bitmessage un logiciel de messagerie chiffrée décentralisé, ...

Mais aussi dans des jeux vidéos :

- Civilization IV utilise Python pour la majorité de ses tâches
- Disney's Toontown Online est écrit en Python
- Battlefield 2 utilise Python pour tous ses addons
- World of Tanks utilise Python pour la majorité de ses tâches, ...

Et est utilisé par de nombreuses entités : la NASA, Google, Youtube, Reddit, Yahoo, EDF, ...

`https://fr.wikipedia.org/wiki/Liste_de_logiciels_Python#Applications`

Deux modes d'exécution

Deux modes d'exécution

Il existe deux modes d'exécution distincts pour utiliser Python :

- Le **mode interprété**
- L'utilisation de **scripts** Python

Mode interprété

Mode interprété

Le mode interprété, appelé aussi *calculatrice Python*, permet l'utilisation interactive de **l'interpréteur**.

Mode interprété

Mode interprété

Le mode interprété, appelé aussi *calculatrice Python*, permet l'utilisation interactive de **l'interpréteur**.

```
[bonzon@pc529-1:~]$ python3
Python 3.6.8 |Anaconda, Inc.| (default, Dec 30 2018, 01:22:34)
[GCC 7.3.0] on linux

>>>
```

Mode interprété

Mode interprété

Le mode interprété, appelé aussi *calculatrice Python*, permet l'utilisation interactive de **l'interpréteur**.

```
[bonzon@pc529-1:~]$ python3
Python 3.6.8 |Anaconda, Inc.| (default, Dec 30 2018, 01:22:34)
[GCC 7.3.0] on linux

>>> 3*54
162
>>>
```

Mode interprété

Mode interprété

Le mode interprété, appelé aussi *calculatrice Python*, permet l'utilisation interactive de **l'interpréteur**.

```
[bonzon@pc529-1:~]$ python3
Python 3.6.8 |Anaconda, Inc.| (default, Dec 30 2018, 01:22:34)
[GCC 7.3.0] on linux

>>> 3*54
162
>>> print("Hello world !")
Hello world !
>>>
```

Mode interprété

Mode interprété

Le mode interprété, appelé aussi *calculatrice Python*, permet l'utilisation interactive de **l'interpréteur**.

```
[bonzon@pc529-1:~]$ python3
Python 3.6.8 |Anaconda, Inc.| (default, Dec 30 2018, 01:22:34)
[GCC 7.3.0] on linux

>>> 3*54
162
>>> print("Hello world !")
Hello world !
>>>
```

- Pour valider une instruction : appuyer sur `Entree`
- Pour quitter l'interpréteur : commande `exit()` ou appuyer sur `Ctrl-D`

Utilisation de scripts Python

Scripts Python

- L'utilisation de l'interpréteur présente vite des limites dès qu'on veut exécuter une suite d'instructions plus complexe
- Enregistrer ces instructions dans un fichier, appelé script Python
- Extension standard : `monfichier.py`

Utilisation de scripts Python

Scripts Python

- L'utilisation de l'interpréteur présente vite des limites dès qu'on veut exécuter une suite d'instructions plus complexe
- Enregistrer ces instructions dans un fichier, appelé script Python
- Extension standard : `monfichier.py`

Deux possibilités pour exécuter le script :

1. Donner le script comme argument à la commande Python
2. Rendre le fichier exécutable

Script donné en argument

1. donner le script comme argument à la commande Python :
   1.1 Créer le fichier `helloworld.py` avec un éditeur de texte

```
print("Hello world !")
```

   1.2 L'exécuter en Python :

```
[bonzon@pc529-1:~]$ python3 helloworld.py
Hello world !
```

Fichier Python exécutable

2. Rendre le fichier exécutable

   2.1 Préciser au `shell` la localisation de l'interpréteur Python dans la première ligne du script

```
#! /usr/bin/env python3
print("Hello world !")
```

   2.2 Rendre le script Python exécutable

```
[bonzon@pc529-1:~]$ chmod +x helloworld.py
```

   2.3 Exécuter le script

```
[bonzon@pc529-1:~]$ ./helloworld.py
Hello world !
```

Les commentaires

Commentaires

Les commentaires permettent d'annoter le code.

Tout ce qui suit le caractère # est ignoré.

Exception : la première ligne peut être
`#! /usr/bin/env python3`
et a alors une signification particulière pour Python !

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

# votre premier script Python
print("Hello world !")

# ----------------------
# D'autres commentaires
# ----------------------
```

Variables

Variable

Variable

Une variable est une **zone de la mémoire** dans laquelle une **valeur** est stockée.

Une variable possède 4 propriétés :

- un nom
- une adresse
- un type
- une valeur

Variable

Variable

Une variable est une **zone de la mémoire** dans laquelle une **valeur** est stockée.

Une variable possède 4 propriétés :

- un nom
- une adresse
- un type
- une valeur

- Pour le programmeur, une variable est définie par son **nom**
- Pour l'ordinateur, c'est une **adresse** de la mémoire

Variable

En Python, la **déclaration** d'une variable et son **initialisation** se font en même temps.

```
>>> x = 2
>>> x
2
>>>
```

Variable

En Python, la **déclaration** d'une variable et son **initialisation** se font en même temps.

```
>>> x = 2
>>> x
2
>>>
```

On illustre ce mécanisme par un cercle qui contient un nom de référence, et un rectangle une valeur :

Variable

En Python, la **déclaration** d'une variable et son **initialisation** se font en même temps.

```
>>> x = 2
>>> x
2
>>>
```

On illustre ce mécanisme par un cercle qui contient un nom de référence, et un rectangle une valeur :

L'attribution d'une valeur à une variable s'appelle **l'affectation**.

Identificateurs

Identificateur

Les noms des variables (et des fonctions) sont appelés des identificateurs.

Règles de formation des identificateurs :

- Peut contenir des lettres (minuscules 'a' ... 'z', majuscules 'A' ... 'Z'), des chiffres ('0' ... '9'), et le caractère de soulignement ('_')
- Le premier caractère doit être une lettre
- Sensible à la casse
  - `test`, `Test` et `TEST` représentent 3 variables différentes
- Ne doit contenir ni espace, ni caractères spéciaux
- Eviter les mots réservés de Python 3

Mots réservés de Python 3

Python 3 contient 33 mots clés :

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

Conventions lexicales

- Un bon programmeur doit veiller à ce que ses lignes d'instructions soient faciles à lire : un identificateur doit être aussi explicite que possible !
- Importance d'utiliser une **politique cohérente** de nommage des identificateurs :
  - **Constantes** : tout en majuscules – `MACONSTANTE`
  - Sinon, tout en minuscules, sauf à l'intérieur du nom pour augmenter la lisibilité – `somme, test, tableDesMatieres`

Affectation d'une valeur à une variable

Affectation

L'affectation est l'opération qui consiste à attribuer une valeur à une variable.

Elle est réalisée grâce à l'opérateur =

```
>>> nb = 42
>>> msg = "Quoi de neuf?"
>>> pi = 3.14157
```

Affectation d'une valeur à une variable

Les valeurs des variables sont mémorisées, on peut les afficher :

```
>>> nb
42
>>> msg
'Quoi de neuf?'
>>> pi
3.14157
```

Affectation d'une expression à une variable

On peut également affecter une expression à une variable.

Expression

Une expression est une portion de code que l'interpréteur Python peut **évaluer** pour obtenir une **valeur**

```
>>> nb2 = nb + pi
>>> nb2
45.14157
>>> titre = "-"*10 + "Titre" + "-"*10
>>> titre
'----------Titre----------'
```

Affectation et égalité

Attention : affecter n'est pas comparer !

Attention de ne pas confondre **affectation** et **égalité mathématique** !

- L'affectation (=) a un **effet** (associe une valeur à une variable), mais n'a pas de **valeur**
- La comparaison (==) a une **valeur** (`True` si la comparaison est vraie, `False` sinon), mais n'a pas d'**effet**

Affectation et égalité

Attention : affecter n'est pas comparer !

Attention de ne pas confondre **affectation** et **égalité mathématique** !

- L'affectation (=) a un **effet** (associe une valeur à une variable), mais n'a pas de **valeur**
- La comparaison (==) a une **valeur** (`True` si la comparaison est vraie, `False` sinon), mais n'a pas d'**effet**

```
>>> x = 3      #la valeur 3 est affectée à x
>>> y = 5      #la valeur 5 est affectée à y
>>> x == y     #est-ce que x est égal à y ?
False
```

Ré-affecter une valeur à une variable

Réaffectation

La valeur d'une variable peut évoluer au cours du temps par des opérations de réaffectation.

Ré-affecter une valeur à une variable

Réaffectation

La valeur d'une variable peut évoluer au cours du temps par des opérations de réaffectation.

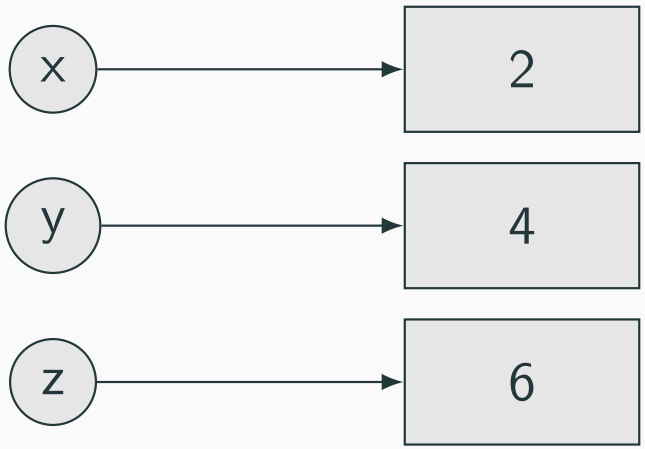
```
>>> x = 2
>>> y = 4
>>> z = 6
```

Ré-affecter une valeur à une variable

Réaffectation

La valeur d'une variable peut évoluer au cours du temps par des opérations de réaffectation.

```
>>> x = 2
>>> y = 4
>>> z = 6
```

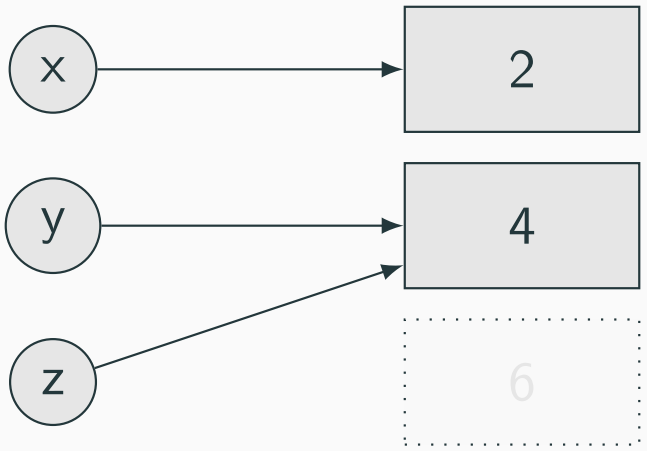


Ré-affecter une valeur à une variable

Réaffectation

La valeur d'une variable peut évoluer au cours du temps par des opérations de réaffectation.

```
>>> x = 2
>>> y = 4
>>> z = 6
>>> z = y
```

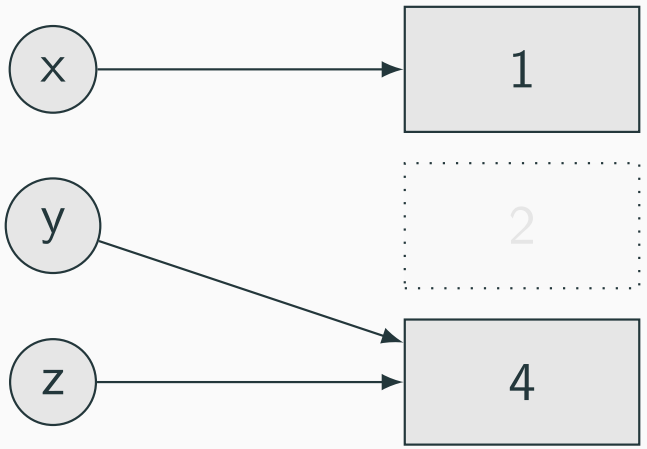


Ré-affecter une valeur à une variable

Réaffectation

La valeur d'une variable peut évoluer au cours du temps par des opérations de réaffectation.

```
>>> x = 2
>>> y = 4
>>> z = 6
>>> z = y
>>> x = 1
```



Séquence temporelle des affectations

Séquence temporelle des affectations

Le **membre droit** d'une affectation est évalué avant de réaliser l'affectation.

La variable affectée peut donc se trouver en partie droite de l'affectation.

```
>>> x = 2
>>> x = x + 1              #incrémentation : on calcule x+1,
>>> x                      #puis on affecte cette valeur à x
3
>>> x = x - 1              #décrémentation : on calcule x-1,
>>> x                      #puis on affecte cette valeur à x
2
>>> u_n = 4
>>> u_n = 3 * u_n + 1      #définition d'une suite numérique
>>> u_n
13
```

Affectations multiples

Affectations multiples

Sous Python, on peut assigner une valeur à plusieurs variables simultanément.

```
>>> x = y = 3    #y reçoit 3, puis x reçoit la valeur de y
>>> x            #Ils référencent la même donnée
3
>>> y
3
```

Affectations parallèles

Affectations parallèles

On peut aussi effectuer des affectations parallèles à l'aide d'un seul opérateur.

```
#affectation parallèle de a à 2, b à 8.3
>>> a, b = 2, 8.3
>>> a
2
>>> b
8.3
```

Affectations parallèles et évaluation d'expressions

Affectations parallèles et évaluation d'expressions

Dans une affectation, la valeur du membre de droite est évaluée avant de l'affecter au membre de gauche, ce qui est important pour les affectations parallèles.

```
>>> a = 3
>>> b, a = a + 2, a * 2
#toutes les expressions sont évaluées avant la 1ère affectation
>>> a
6
>>> b
5
```

Suppression d'une variable

Suppression d'une variable

Il est possible au cours de l'exécution de supprimer une variable (et donc un nom qui référence une donnée).

L'instruction qui permet cela est `del`.

```
>>> a = 3
>>> a
3
>>> del a
>>> a
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
```

Les types de données

Les types de données

Types

Le type d'un objet Python détermine de quelle sorte d'objet il s'agit.

La fonction `type()` fournit le type d'une valeur

Les types de données

Types

Le type d'un objet Python détermine de quelle sorte d'objet il s'agit.

La fonction `type()` fournit le type d'une valeur

Les principaux types de données en Python sont :

- les entiers, type `int`
- les réels (ou flottants), type `float`
- les booléens, type `bool`
- les chaînes de caractères, type `str`
- les listes, type `list`
- les tuples (ou n-uplets), type `tuple`
- les dictionnaires, type `dict`
- les ensembles, type `set`

Les types en Python

Typage des variables

Le typage des variables sous Python est un typage dynamique.

- Inutile de définir manuellement le type des variables en Python
- Il suffit d'assigner une valeur à un nom de variable pour que celle ci soit automatiquement créée avec le type qui correspond le mieux à la valeur fournie

```
>>> nb = 42
>>> msg = "Quoi de neuf?"
>>> pi = 3.14157
>>> type(nb)
<class 'int'>
>>> type(msg)
<class 'str'>
>>> type(pi)
<class 'float'>
```

Les entiers, le type `int`

Les opérations sur les entiers

- l'opposé (opération unaire, notée -)
- l'addition (opération binaire, notée +)
- la soustraction (opération binaire, notée -)
- la multiplication (opération binaire, notée *)
- la puissance (opération binaire, notée **)
- la division (opération binaire, notée /)
- la division entière (opération binaire, notée //)
- le reste de la division entière (opération binaire, notée %)

Attention la multiplication n'est pas implicite, le symbole * doit toujours être indiqué explicitement entre les deux opérandes.

Les opérations sur les entiers

Opération	Valeur
2 + 3	5
2 - 3	-1
2 * 3	6
2 ** 3	8
17 / 5	3.4
17 // 5	3
17 % 5	2

Les réels, le type `float`

type `float`

Un `float` est noté avec un point décimal (**jamais une virgule**), ou en notation exponentielle avec le symbole `e` ou `E` symbolisant le *10 puissance* suivi des chiffres de l'exposant.

Les flottants supportent les mêmes opérations que les entiers.

Les réels, le type `float`

type `float`

Un `float` est noté avec un point décimal (**jamais une virgule**), ou en notation exponentielle avec le symbole `e` ou `E` symbolisant le *10 puissance* suivi des chiffres de l'exposant.

Les flottants supportent les mêmes opérations que les entiers.

Opération	Valeur
2 e 3	2000.0
-1.6 e -2	-0.016

Ordre de priorité des opérateurs

PEMDAS

1. P : parenthèses
2. E : exposant
3. MD : multiplication et division
4. AS : additions et soustractions

A priorité égale, les opérations sont évaluées de la gauche vers la droite

Opération	Valeur
$5 + 4 * 2$	13
$(5 + 4) * 2$	18

Les chaines de caractères, le type `str`

Les opérations sur les chaines de caractères

- La concaténation, opérateur `+`
- La duplication, opérateur `*`
- La longueur, opérateur `len`
- Le test d'appartenance, opérateur `in`
- l'utilisation de l'apostrophe (`'`) à la place du guillemet (`"`) autorise l'inclusion d'une notation dans l'autre

Les chaines de caractères, le type `str`

```
>>> chaine = "Salut "
>>> chaine + "Python"    #concaténation
'Salut Python'
>>> chaine*3             #duplication
'Salut Salut Salut '
>>> "thon" in "Python"   #test d'appartenance
True
>>> len("Salut Python") #longueur
12
>>> phrase = 'Il a dit "Bonjour"' #apostrophes et guillemets
>>> phrase
'Il a dit "Bonjour"'
```

Entrées/Sorties

Les entrées

La saisie

La fonction `input()` permet d'affecter à une variable une valeur tapée sur le clavier.

La valeur retournée est toujours du type str, mais on peut ensuite changer le type (on dit aussi transtyper).

```
>>> i = input("Entrez un entier : ")
Entrez un entier : 4
>>> type(i)                    #i est une chaine de caractère
<class 'str'>
>>> i = int(i)                 #i est maintenant un entier
>>> type(i)
<class 'int'>
```

Le transtypage

Python est un langage dynamique (les variables peuvent changer de type), mais fortement typé (il y a un contrôle de la cohérence des types).

```
>>> i = input("Entrez un entier : ")
Entrez un entier : 4
>>> type(i)
<class 'str'>                #i est une chaine de caractère
>>> i = int(input("Entrez un entier : "))
Entrez un entier : 4
>>> type(i)
<class 'int'>                #i est un entier
>>> iplus = int(input("Entrez un entier : ")) + 1
Entrez un entier : 3
>>> iplus
4
>>> ibug = input("Entrez un entier : ")) + 1
File "<stdin>", line 1
ibug = input("Entrez un entier : ")) + 1
^
SyntaxError: invalid syntax
```

Les sorties

L'écriture

La fonction `print()` permet d'afficher la représentation textuelle des informations qui lui sont données en paramètre.

```
>>> chaine = "Bonjour Python"
>>> chaine
'Bonjour Python'
>>> print(chaine)
Bonjour Python
>>> a, b = 2, 5
>>> print('Somme :', a + b, 'et', a - b, 'est la différence.')
Somme : 7 et -3 est la différence.
```

Les séquences d'échappement

Séquences d'échappement

A l'intérieur d'une chaîne de caractères, le caractère antislash (\) permet de donner une signification spéciale à certaines séquences de caractères

Séquence	Signification
\	saut de ligne ignoré (placé en fin de ligne)
\\	antislash
\'	apostrophe
\"	guillemet
\n	saut de ligne
\f	saut de page
\r	retour en début de ligne
\t	tabulation horizontale
\v	tabulation verticale

Les séquences d'échappement : exemple

```
#Retour à la ligne
>>> print("Heureux qui, comme Ulysse,\n a fait un beau voyage")
Heureux qui, comme Ulysse,
 a fait un beau voyage
#Retour au début de ligne (et écrasement)
>>> print("Heureux qui, comme Ulysse,\r a fait un beau voyage")
a fait un beau voyagesse,
#tabulation horizontale
>>> print("Heureux qui, comme Ulysse,\t a fait un beau voyage")
Heureux qui, comme Ulysse,       a fait un beau voyage
#tabulation verticale
>>> print("Heureux qui, comme Ulysse,\v a fait un beau voyage")
Heureux qui, comme Ulysse,
                          a fait un beau voyage
```

Pour conclure

Résumé du cours

Aujourd'hui, on a vu

- L'organisation du cours
- Ce qu'est un langage compilé, interprété, semi-interprété
- Comment utiliser la calculatrice Python, un script Python
- Comment écrire des commentaires dans un script Python
- Ce qu'est une variable, comment on la nomme, comment on la déclare
- Ce qu'est une affectation, les affectations multiples ou parallèles
- Quelques types de données (les entiers, les réels, les chaines de caractères)
- Les entrées/sorties