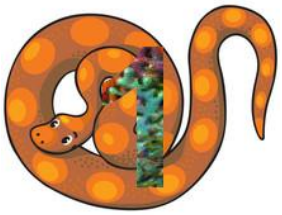
Python 3 -TDs- Fipa1



Saison 1 - épisode 1

1 Découvrons et testons le langage Python

1.1 Les modes d'exécution

Python présente la particularité de pouvoir être utilisé de plusieurs manières différentes.

Vous allez l'utiliser en ***mode interactif***, c'est-a-dire d'une manière telle que vous pourrez dialoguer avec lui directement depuis le clavier. Cela vous permettra de découvrir très vite un grand nombre de fonctionnalités du langage. Dans un second temps, vous apprendrez comment créer vos premiers programmes (scripts) et les sauvegarder sur disque.

Interpréteur Python : exécution d'une boucle d'évaluation d'instructions

Script Python : enregistrement des instructions dans un fichier grâce à un éditeur.

Vous pouvez tout de suite utiliser l'**interpréteur** comme une simple calculatrice.

1.2 Identificateurs et mots clés

Les identificateurs

Python utilise des identificateurs, suite non vide de caractères, pour nommer ses objets.

Il existe des règles de nommage : pour nommer les éléments du langage (minuscule, majuscule, _ …)

nom_de_ma_variable	UneExceptionError
NOM_DE_MA_CONSTANTE	Nom_de_module
maFonction, maMethode	MaClasse

L'***identificateur*** ne peut être un ***mot réservé*** de Python (and, as, assert, break, class, continue …)

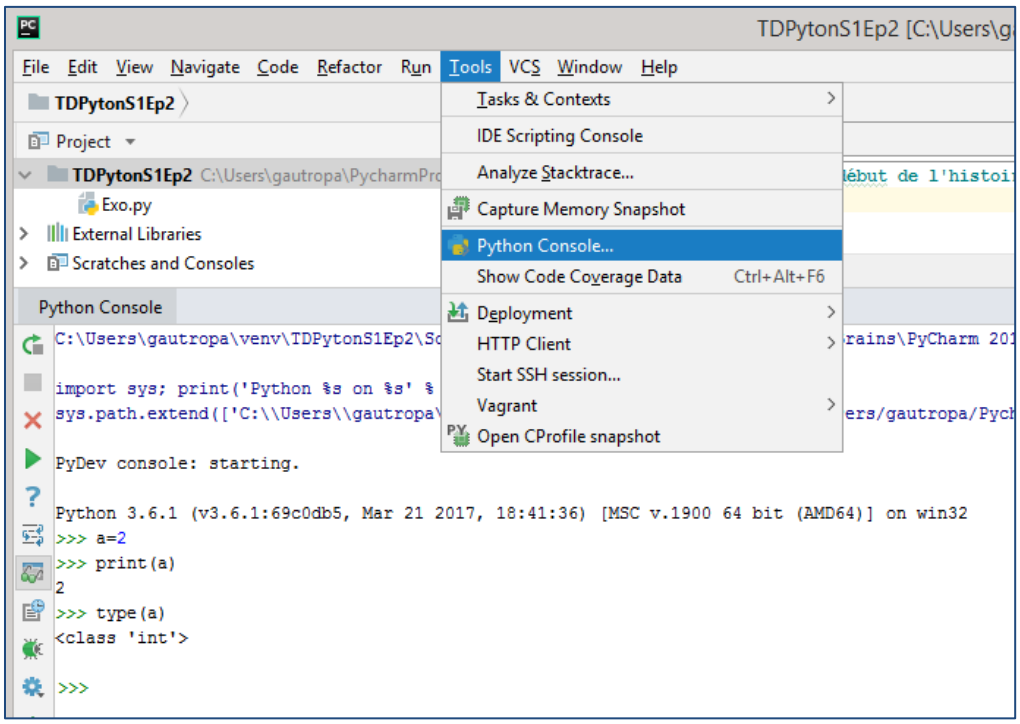
Les mots réservés pour Python

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

Les expressions sont une combinaison **d'*identificateurs*** : id_exo **; d'*opérateurs :*** !=, = ***;*** **de *littéraux*** : 8, 6

Testons l'Interpréteur Python avec Pycharm **!**

En mode Console voir menu Tools>Python Console



Testons quelques opérations !

```
Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 18:41:36) [MSC v.1900 64 bit (AMD64)]

Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 18:41:36) [MSC v.1900 64 bit (AMD64)] on win32
In[2]: 9+4
Out[2]: 13
In[3]: 9+4*2
Out[3]: 17
In[4]: 9 + (4*2)
Out[4]: 17
In[5]: (9+4) *2
Out[5]: 26
In[6]: 20/4
Out[6]: 5.0
In[7]: 20/3
Out[7]: 6.666666666666667
In[8]: 20//3
Out[8]: 6
In[9]: 20.8/3
Out[9]: 6.933333333333334
In[10]: 20,8/3
Out[10]: (20, 2.6666666666666665)
```

```
In[12]: 8/3
Out[12]: 2.6666666666666665

In[13]:
```

1.3 Les types de données entiers

1.3.1 Le type int

Limité en taille par la mémoire de la machine

```
>>> 2013          # décimal (base 10)          2013
>>> 0b11111011101    # binaire (base 2)        2013
>>> 0o3735        # octal (base 8)             2013
>>> 0x7dd         # hexadécimal (base 16)      2013
```

```
In[2]: 2013
Out[2]: 2013
In[3]: 0b11111011101
Out[3]: 2013
In[4]: 0o3735
Out[4]: 2013
In[5]: 0x7dd
Out[5]: 2013
```

```
In[8]: 2*3
Out[8]: 6
In[9]: 2**3
Out[9]: 8
```

Les opérations arithmétiques

+	addition
-	soustraction
*	multiplication
**	puissance
/	division flottante entre 2 entiers 20/3 => 6,6666666667
//	division entière 20//3 => 6
%	modulo

Les bases arithmétiques

En arithmétique, une base *n* désigne la valeur dont les puissances successives interviennent dans l'écriture des nombres, ces puissances définissant l'ordre de grandeur de chacune des positions occupées par les chiffres composant tout nombre.

Par exemple : $57n = (5xn^1) + (7xn^0)$

Certaines bases sont couramment employées :

- la base 2 (système binaire), en électronique numérique et informatique ;
- la base 8 (système octal), en informatique ;
- la base 16 (système hexadécimal), fréquente en informatique ;
- la base 60 (système sexagésimal), dans la mesure du temps et des angles.

Les changements de base

Un nombre dans une base *n* donnée s'écrit sous la forme d'addition des puissances successives de cette base.

Exemples

$57_{16} = (5x16^1) + (7x16^0) = 87_{10}$

$57_8 = (5x8^1) + (7x8^0) = 47_{10}$

Un entier écrit en base 10 (par exemple 179) peut être représenté en

binaire

```
In[10]: 0b10110011
Out[10]: 179
In[11]: bin(179)
Out[11]: '0b10110011'
```

octal

```
In[14]: 0o263
Out[14]: 179
In[15]: oct(179)
Out[15]: '0o263'
```

hexadécimal

```
In[16]: 0xB3
Out[16]: 179
In[17]: hex(179)
Out[17]: '0xb3'
```

1.3.2 Le type bool

2 valeurs possibles : **True, False**

Les opérateurs produisant un résultat de type bool :

opérateurs de comparaison	
== **!=** **> et >=** **< et <=**	
opérateur unaire	**not**
opérateur binaire	**and** **or**

Opérateurs logiques : not, or, and

Dès qu'un premier membre a la valeur False, l'expression est False. La deuxième expression n'est pas évaluée. Idem pour True. Optimisation appelée « principe du ***shortcut*** »

Testons !

```
In[18]: 56>100
Out[18]: False
In[19]: 56<100
Out[19]: True
In[20]: 56<=100
Out[20]: True
```

```
In[22]: 6==6
Out[22]: True
In[23]: (5==5) or (7>5)
Out[23]: True
In[24]: (8>10) and (5==5)
Out[24]: False
```

1.4 Les types de données flottants

En informatique on utilise une représentation interne pour exprimer la notion mathématique de réel. Ces nombres sont appelés des flottants.

1.4.1 Le type float (flottants)

Vous avez déjà rencontré précédemment cet autre type de donnée numérique : le type « nombre réel », ou « nombre à virgule flottante », désigné en anglais par l'expression *floating point number*, et que pour cette raison on appellera type *float* sous Python. Ce type autorise les calculs sur de très grands ou très petits nombres avec un degré de précision constant. Pour qu'une donnée numérique soit considérée par Python comme étant du type *float*, il suffit qu'elle contienne dans sa formulation un élément tel qu'un point décimal ou un exposant de 10.

Par exemple, les données suivantes sont automatiquement interprétées par Python comme étant du type float :

3.14	10.	.001	1e100	3.14e-10

Un ***float*** est noté avec un point décimal, ou en notation exponentielle avec un « e » symbolisant le « 10 puissance » suivi des chiffres de l'exposant.

Exemple : 2.657, .06, $-1.7^{e}-12$

```
In[25]: 2.657
Out[25]: 2.657
In[26]: -1.7**-12
Out[26]: -0.0017163780166626966
```

Attention !

```
In[28]: -1.7**-2
Out[28]: -0.34602076124567477
In[29]: -1.7e-2
Out[29]: -0.017
```

Les flottants supportent les mêmes opérateurs que les entiers. Ils ont une précision finie limitée.

1.4.2 Le type complex

Les complexes sont écrits en notation cartésienne formée de deux flottants. La partie imaginaire est suffixée par j :

```
>>>
>>> import cmath
>>> print(cmath.phase(-1+0j)
... )
3.141592653589793
>>> print(1j)
1j
>>> print((2+3j) + (4-7j))
(6-4j)
>>> print((9+5j).real)
9.0
>>> print((9+5j).imag
... )
5.0
>>>
```

1.5 Données, variables et affectation

Le travail d'un programme informatique consiste pour l'essentiel à manipuler des ***données***. Ces données de différents types s'avèrent n'être qu'une suite finie de nombres binaires dans la mémoire de l'ordinateur. Les données sont enregistrées et accessibles en mémoire vive le temps de l'exécution du programme.

Pour manipuler les données, un programme stocke chaque donnée à une adresse mémoire précise. Une ***variable*** repérée par un nom correspond à cette référence. Les variables permettent de manipuler des données de différents types stockées à un emplacement précis en mémoire.

Attention : Ne pas confondre affectation (=) et égalité (==) !

Testons !

Les variantes de l'affectation :

```
>>> a=2
>>> a=a+1
>>> a
3
>>> a=a-1
>>> a
2
```

```
>>> a = 3*9
>>> a
27
>>> b=a*2
>>> b
54
>>> a= b/2
>>> a
27.0
>>> c=b/6
>>> c
9.0
>>> a=c*4
>>> a
36.0
```

```
>>> a = 5
>>> a += 2
>>> a
7
>>> b=a
>>> b
7
>>> c= d = 8
>>> c,d
(8, 8)
>>> e,f = 1.4, 3.8
>>> e,f
(1.4, 3.8)
>>> a = 6
>>> a,b = a+2, a*7
>>> a,b
(8, 42)
```

Priorité des expressions

Dans une expression les parenthèses ont la plus haute priorité. Elles permettent de forcer l'évaluation d'une expression dans l'ordre choisi.

```
>>> 2*(6-2)
8
>>> (2*6)-2
10
```

```
>>> 2**1+1
3
>>> 2**(1+1)
4
```

1.6 Les données alphanumériques

Un programme d'ordinateur peut traiter des caractères alphabétiques, des mots, des phrases, ou des suites de symboles quelconques. Dans la plupart des langages de programmation, il existe pour cet usage des structures de données particulières que l'on appelle ≪ chaines de caractères ≫.

Une donnée de type **string** ou **str** peut se définir comme une suite quelconque de caractères. En Python, on délimitera une telle suite de caractères par des guillemets (double quotte) ou des apostrophes (simple quotte).

Testons !

```
>>> expression1 = 'le jour'
>>> expression2 = "se levera"
>>> expression3 = 'encore'
>>> expression1
'le jour'
>>> expression1+expression2+expression3
'le jourse leveraencore'
>>> expression2 = " se lèvera "
>>> expression1+expression2+expression3
'le jour se lèvera encore'
```

Les trois variables sont de type ***string***

```
>>> type(expression2)
<class 'str'>
```

La fonction ***print()*** insère un espace entre les les éléments à afficher !

```
>>> expr1='le'
>>> expr2='genre'
>>> expr3="féminin"
>>> print(expr1,expr2,expr3)
le genre féminin
```

1.6.1 Les séquences d'échappement

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

Séquence	Signification
\\	affiche un antislash
\'	apostrophe
\"	guillemet
\a	sonnerie (*bip*)
\b	retour arrière
\f	saut de page
\n	saut de ligne
\r	retour en début de ligne
\t	tabulation horizontale
\v	tabulation verticale
\N{nom}	caractère sous forme de code Unicode nommé
\uhhhh	caractère sous forme de code Unicode 16 bits
\Uhhhhhhhh	caractère sous forme de code Unicode 32 bits
\ooo	caractère sous forme de code octal
\xhh	caractère sous forme de code hexadécimal

Testons !

```
>>> expr1='le'
>>> expr2=' genre humain \n'
>>> expr3='est il pacifique ?'
>>> print(expr1,expr2,expr3)
le  genre humain
 est il pacifique ?
```

```
>>> print("d \144 \x64")
d d d
>>> print(r"d \144 \x64")
d \144 \x64
```

1.6.2 Trois Opérations sur les chaines de caractères

Longueur => *lens(s)*

Concaténation => *s1 = s2 + s3*

Répétition => *s1 = s2*s3*

Testons !

```
>>> s= "DoubidoubidouOuha"
>>> len(s)
17
>>> s1=" Hello "
>>> s3 = " ! "
>>> s2= s + s1 +s3
>>> s2
'DoubidoubidouOuha Hello  ! '
>>> s= s+s3
>>> s
'DoubidoubidouOuha ! '
>>> s4= s1 * 6
>>> s4
' Hello  Hello  Hello  Hello  Hello  Hello '
>>>
```

1.6.3 Fonctions et méthodes

Les chaines de caractères sont manipulables en utilisant des fonctions (notion procédurale) communes à tous les types de séquences ou bien des méthodes (notion objet) spécifiques aux chaines.

Fonction => ***len()*** et méthode => ***upper()***

```
>>> l=len("le soleil brille")
>>> l
16
>>> "le soleil brille".upper()
'LE SOLEIL BRILLE'
>>>
```

Méthodes retournant une valeur booléenne **: isupper(), istitle(), isalnum(), startswith(), endswith()**

Méthodes retournant une nouvelle chaine **: lower(), upper(), capitalize() et swapcase()**

- **expandtabs([tabsize]) :** remplace les tabulations par tabsize espaces (□ par défaut).
- **center(width[, fillchar]), ljust(width[, fillchar])** et**rjust(width[, fillchar]) :** retournent respectivement une chaîne centrée, justifiée à gauche ou à droite, complétée par le caractère fillchar (ou par l'espace par défaut)
- **zfill(width) :** complète ch à gauche avec des 0 jusqu'à une longueur maximale de width
- **strip([chars]), lstrip([chars]) et rstrip([chars]) :** suppriment toutes les combinaisons de chars (ou l'espace par défaut) respectivement au début et en fin, au début, ou en fin d'une chaîne
- **f ind(sub[, start[, stop]]) :** renvoie l'index de la chaîne sub dans la sous-chaîne start à stop, sinon renvoie . rfind() effectue le même travail en commençant par la fin. index() et rindex() font de même mais produisent une erreur (exception) si la chaîne n'est pas trouvée
- **replace(old, new[, count])** : remplace count instances (toutes par défaut) de old par new
- **split(seps[, maxsplit]) :** découpe la chaîne en maxsplit morceaux (tous par défaut). rsplit() effectue la même chose en commençant par la fin et striplines() effectue ce travail avec les caractères de fin de ligne**.**
- **join(seq) : concatène les chaînes du conteneur seq en intercalant entre chaque élément la chaîne sur laquelle** la méthode est appliquée**.**

Testons !

```
>>> s.startswith('T')
False
>>> s.endswith('ha')
False
>>> s.endswith('ha ! ')
True
>>> s.lower()
'doubidoubidououha ! '
>>> s.upper()
'DOUBIDOUBIDOUOUHA ! '
>>> s.capitalize()
'Doubidoubidououha ! '
>>> s.swapcase()
'dOUBIDOUBIDOUoUHA ! '
```

```
In[75]: b="Bonjour la lune"
In[76]: c=b.split("la")
In[77]: print(c)
['Bonjour ', ' lune']
```

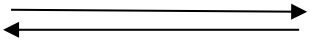
1.6.4 Indexation simple

Pour indexer une chaine de caractère, on utilise l'opérateur **[]**.

Soit s ='Doubidouba'

Len(s) ➔ 10

s[0], s[1], s[2] ….. s[8] …..



s[-8], s[-7], s[-6] ….. s[-1]

Testons !

```
>>> s='Doubidouba'
>>> len(s)
10
>>> s[0]
'D'
>>> s[1]
'o'
>>> s[2]
'u'
>>> s[9]
'a'
>>> s[10]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> s[-1]
'a'
>>> s[-2]
'b'
>>> s[-7]
'b'
>>> s[-8]
'u'
>>> s[-9]
'o'
>>> s[-10]
'D'
```

Extraction de sous-chaines :

Avec 2 ou 3 index séparés par le caractère « **:** », l'opérateur « **[]** » permet d'extraire des sous-chaines

```
>>> s[1:4]
'oub'
>>> s[4:]
'idouba'
>>> s[:7]
'Doubido'
>>> s[::-1]
'abuodibuoD'
>>> s[::3]
'Dboa'
>>> s[:3]
'Dou'
>>> s[-6:]
'idouba'
>>>
```

1.7 Fonctions prédéfinies du langage

Les flux d'entrée sortie : **input()** pour saisir une donnée d'entrée
print() pour afficher la valeur des données

Testons !

```
>>> a = input("Entrez une valeur pour a :")
Entrez une valeur pour a : 12
>>> print(a)
 12
>>>
```

```
>>> a = input("Entrez une valeur :")
Entrez une valeur :4
>>> b = input("Entrez une autre valeur : ")
Entrez une autre valeur : 5
>>> print(a+b)
45
>>> int(a)
4
>>> int(b)
5
>>> print(a+b)
45
>>> type(a)
<class 'str'>
>>> type(b)
<class 'str'>
```

```
<class 'int'>
>>> a=int(input("Entrez un nombre : "))
Entrez un nombre : 3
>>> type(a)
<class 'int'>
>>> b=int(input("Entrez un nombre : "))
Entrez un nombre : 4
>>> type(b)
<class 'int'>
>>> print(a+b)
7
```

```
>>> a,b = 4,5
>>> print(a,b)
4 5
>>> print ( "La valeur ",a, " est inférieure à la valeur ",b)
La valeur  4  est inférieure à la valeur  5
>>> print ( "La valeur ",a, " est inférieure à la valeur ",b, sep="#")
La valeur #4# est inférieure à la valeur #5
>>> print ( "La valeur ",a, " est inférieure à la valeur ",b, sep="#",
end="@@@")
La valeur #4# est inférieure à la valeur #5@@@>>>
```

2 Mise en pratique

Exercice 1

Calculez le volume d'un cône droit à partir d'un rayon et d'une hauteur.

(v= (pi*r^2*h)/3)

Exercice 2

A partir de la saisie d'un prix HT, et du montant d'une TVA calculez le prix TTC d'un produit de consommation.

Exercice 3

Tracez une ligne entre 2 points représentant 2 villes situées à une certaine distance l'une de l'autre en utilisant le – , le x, l'opérateur sep et 2 noms de ville :

```
Brest x-----------------------------x Quimper
```

Exercice 4

```
Combien de marches ? 100
Hauteur d'une marche (cm) ? 20
hauteur parcourue : 1400.0
```

Un gardien de phare doit régulièrement redescendre au rez-de-chaussée pour s'acquitter de tâches de surveillance d'alarmes.

On estime à 5 le nombre moyen d'allers-retours quotidiens.

Les deux paramètres, le nombre x de marches du phare et la hauteur y de chaque marche seront saisis (en cm). Affichez le résultat comme ci-dessous :

« Pour x marches de y cm, il parcourt z.zz m par semaine ».

```
Pour 100 marches de 20 cm, il parcourt par semaine 1400.000000 mètres
```

On n'oubliera pas :

— qu'une semaine comporte 7 jours ;

— qu'une fois en bas, le gardien doit remonter ;

— que le résultat est à exprimer en m.

Exercice 5

Ecrivez un programme qui convertit un nombre entier de secondes fourni au départ en un nombre d'années, de mois, de jours, de minutes et de secondes (utilisez l'opérateur modulo : %). (Testez avec un grand nombre comme : 12345678912)

Exercice 6

Ecrivez un programme qui convertisse en radians un angle fourni au départ en degrés, minutes, secondes.

Puisque la circonférence vaut 2 pi R, un angle de 1 radian correspond à 360° / 2 pi ou encore à 180° / pi

Exercice 7

Ecrivez un programme qui convertisse en degrés Celsius une température exprimée au départ en degrés Fahrenheit, ou l'inverse.

La formule de conversion est : T F=TC ×1,8+32.

Exercice 8

Ecrivez un programme qui convertisse en mètres par seconde et en km/h une vitesse fournie par l'utilisateur en miles/heure. (*1 mile = 1609 mètres*)