BCPST 1.1 & 1.2

Aide mémoire Python

1 Les opérations de base

1.1 Opérations dans les programmes

Exposant avec **	5**2 renvoie 25

Affectation avec le signe =
Affectation multiple avec des virgules
Échange de deux valeurs
Incrémentation
	`x = 0`
	`x, y = 0, 1`
	`x, y = y, x`
	`x = x + 1`

Division euclidienne : quotient avec // Division euclidienne : reste avec % Test de parité d'un nombre	20//3 renvoie 6 20%3 renvoie 2 (x\%2)==0 renvoie `True` si x est pair
Affichage avec `print`	`print('Score =', total)`

Saisie au clavier avec `input`
 	
Saisie d'un nombre : conversion de x
	`x=input('votre nom ?')`
	attention x est une chaine de caractères
	`x=int(input('votre age ?'))`

Clic de souris dans une image
 	
Clic dans l'image d'un tableau
	`x,y = ginput(1)[0]`
	attention ce sont les coordonnées cartésiennes du clic
	`x,y = ginput(1)[0]`
	`ligne, colonne = round(y),round(x)`

1.2 Opérations au clavier

CTRL S	sauvegarde du fichier de préférence dans le dossier P :/Travail/TP INFO
CTRL E	provoque l'exécution du fichier attention : il faut que l'exécution se fasse sans message d'erreur pour que les fonctions définies dans le fichier soient ensuite utilisables.
Flèche vers le haut sur le clavier	rappel de la dernière instruction tapée dans le shell
Éclair jaune dans Pyzo	interruption de l'exécution d'un programme à cause d'une boucle `while` infinie en général

2 Les types de variables

Il existe un ensemble de variables avec des propriétés différentes définies dans le langage Python de base. Certains modules (par exemple le module `numpy`) définissent de nouvelles variables avec leurs propriétés. Nous présentons dans ce chapitre les variables de base et ne revenons que sur les propriétés principales.

2.1 Les entiers

Il s'agit du type integer (`type(1)` renvoie `int`).

Python utilise autant de mémoire que nécessaire pour stocker les valeurs exactes des entiers. Il ne fait pas d'arrondis et n'utilise pas la notation scientifique pour les afficher.

```
>>> print(10**30+1)
1000000000000000000000000000001
```

Principales opérations	+, -, *, **
Division euclidienne : quotient avec // Division euclidienne : reste avec % Test de parité d'un nombre	20//3 renvoie 6 20%3 renvoie 2 (x%2) == 0 renvoie `True` si x est pair
Conversion d'un réel en un entier Conversion d'une chaine de caractères en un entier Arrondi d'un réel à l'entier le plus proche avec `round`	`int(3.14)` renvoie 3 `int('18')` renvoie 18 `round(2.83)` renvoie 3

2.2 Les réels

Il s'agit du type float (`type(3.14)` renvoie `float`).

Python utilise la notation scientifique pour stocker les réels sur 64 bits de mémoire. Python fait donc un arrondi à la 16ème décimale : les entiers sont définis avec une précision limitée ! Ainsi :

```
>>> print(10**2+0.1)
100.1
>>> print(10**20+0.1)
1e20
```

Principales opérations La présence d'un point suffit à indiquer qu'il s'agit d'un réel L'opération / renvoie toujours un nombre codé en float Conversion d'une chaine de caractères en un réel	+, -, *, /, ** `type(2.)` renvoie float 6/3 renvoie 2. qui est un float `float('3.14')` renvoie 3.14

2.3 Les booléens

Il s'agit du type booléen noté bool (`type(True)` renvoie `bool`). Il n'y a que deux valeurs possibles : `True` et `False`. Ne pas oublier la majuscule et ne pas mettre de guillemets (ce n'est pas une chaine de caractères).

Lorsque Python fait un test, il renvoie un booléen.

```
>>> print(1<2)
True
>>> print(2<1)
False
```

Principales opérations `and`, `or` et `not`	`True and False` renvoie `False`
	`True or False` renvoie `True`
	`not True` renvoie `False`

Tests classiques ==, >, <, <=, >=
Test d'inégalité !=
Encadrements
	
	`5 != 3+2` renvoie `False`
	`0<5<7` renvoie `True`

Appartenance à une liste : `in` Appartenance à un chaine de caractères : `in`	`3 in [5, 2, 3, 6]` renvoie `True` `'a' in 'bonjour'` renvoie `False`

2.4 Les chaines de caractères

Il s'agit du type string (`type('bonjour')` renvoie `str`). Les chaines de caractères sont définies entre guillemets ou entre apostrophes.

```
>>> x = 'bonjour'
>>> y = "au revoir"
```

Principales opérations :
concaténation : +
longueur de la chaine : len
chaine de caractères vide
	
	`'bon' + 'jour'` renvoie `'bonjour'`
	`len('bonjour')` renvoie 7
	`x = ''`

Chaque caractère est repéré par un indice
dont la numérotation commence à 0
Extraction d'une sous-chaine avec des crochets
`[départ inclu:fin exclue:pas]`
par défaut : départ = 0, fin = `len(chaine)`
pas = 1
	`mot = 'bonjour'`
	`mot[2]` renvoie `'n'`
	`mot[2:4]` renvoie `'nj'`
	`mot[2:7:2]` renvoie `'no'`
	`mot[2::2]` renvoie `'nor'`
	`mot[2:]` renvoie `'njour'`

Appartenance à une chaine de caractères : `in`	`'a' in 'bonjour'` renvoie `False`
Saisie au clavier avec `input` Saisie d'un nombre : conversion de x	`x=input('votre nom ?')` # `type(x) = str` `x=int(input('votre age ?'))` # `type(x) = int`

Attention, à la différence des listes, il n'est pas possible de modifier un caractère d'une chaine de caractères. Ainsi :

```
>>> x = 'bonjour'
>>> x[3] = 't'
Traceback (most recent call last):
  File "<console>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Il faut redéfinir la variable en concaténant le début de la chaine, le nouveau caractère et la fin de la chaine :

```
>>> x = x[:3] + 't' + x[4:]
>>> print(x)
'bontour'
```

2.5 Les listes

Il s'agit du type list (`type([5,2,0])` renvoie `list`). Les listes sont définies entre crochets et les éléments sont séparés par des virgules. Les éléments peuvent être de types différents.

```
>>> L = [4, 'bonjour', True, 0]
>>> print(type(L))
<class 'list'>
```

Chaque caractère est repéré par un indice
dont la numérotation commence à 0
Extraction d'une sous-liste avec des crochets
`[départ inclu:fin exclue:pas]`
par défaut : départ = 0, fin = `len(L)`
pas = 1
	`L=[5, 0, 4, 3, 2, 6]`
	`L[2]` renvoie 4
	`L[2:4]` renvoie `[4, 3]`
	`L[1:5:2]` renvoie `[0,3]`
	`L[2::2]` renvoie `[0, 3, 6]`
	`L[2:]` renvoie `[4, 3, 2, 6]`

Principales opérations :
- répétition : *n
- longueur de la liste : `len`
- liste vide
- concaténation : +
–> ajout d'un élément au début
–> ajout d'un élément à la fin
ou avec `append` (plus rapide)
–> suppression de l'élément d'indice n :
ou avec `pop(n)` (plus rapide)
	
	`[0]*5` renvoie `[0, 0, 0, 0, 0]`
	`len([4, 2, 1]` renvoie 3
	`L = []`
	`L = L1 + L2`
	`L = [5] + L` ajoute 5 au début de `L`
	`L = L + [5]` ajoute 5 à la fin de `L`
	`L.append(5)` – attention syntaxe différente !
	`L = L[:n] + L[n+1:]` enlève l'élément d'indice `n`
	`L.pop(n)` – attention syntaxe différente !

Appartenance à une liste : `in`	`4 in [5, 0, 4, 3, 2, 6]` renvoie `True`
Création d'une liste par compréhension	`L = [i**2 for i in range(4)]` renvoie `[0, 1, 4, 9]`

Quelques précautions avec le parcours des listes :

– Ne pas utiliser de `for` pour parcourir une liste dont la taille varie. En effet dans l'instruction `for i in range(len(L))`, la valeur de `len(L)` n'est évaluée qu'une seule fois (lors du premier passage).

Si la liste est raccourcie dans la boucle, il va y avoir un message d'erreur `'out of range'` (l'indice i sera supérieur au dernier indice de la liste L raccourcie).

Si la liste est rallongée dans la boucle, elle ne sera pas parcourue en entier.

Conclusion : il faut utiliser un `while` pour parcourir une liste de taille variable.

– lorsque l'on parcourt une liste à l'aide d'une boucle `while` il faut tester l'indice (`i < len(L)`) avant de tester la valeur de l'élément.

Exemple pour la recherche du premier zéro avec suppression des éléments négatifs de L qui précédent le premier zéro :

```
def analyse_liste(L):
    """entrée : liste de nombres
    sortie : indice du 1er zéro et liste débarrassée des nb négatifs précédents le 1er zéro"""
    i = 0
    while i < len(L) and L[i]!=0: # test sur i avant test sur L[i]
        if L[i] < 0:
            L.pop(i)
        else :
            i = i+1
    if i == len(L): # on vérifie si on est arrivé au bout de la liste
        return -1,L
    else :          # sinon on a trouvé le 1er zéro
        return i,L
```

2.6 Les tuples

Il s'agit du type tuple (`type( (5,2,0) )` renvoie `tuple`). Les tuples sont définies entre parenthèses et les éléments sont séparés par des virgules. Les éléments peuvent être de types différents.

Attention : les parenthèses du tuple et celle d'une fonction conduisent à un double parenthésage.

```
n = len((2,6,8,9))
```

La principale différence avec les listes est qu'un tuple n'est pas modifiable (tout comme les chaines de caractères). Nous réserverons l'utilisation des tuples aux coordonnées d'une case dans un tableau car la syntaxe suivante est très lisible :

```
pion = (5,2) # coordonnées du pion dans le tableau de jeu
if Tableau[pion]==0:
    print('perdu')
```

2.7 Les dictionnaires

Il s'agit du type dictionnaire (`type(dico)` renvoie `dict`). Les dictionnaires sont définis entre accolades : un élément du dictionnaire est défini par une clé et une valeur séparées par deux points : `clé : valeur` .

```
>>> Dico = {'Bio1':'poireau', 'Bio2':'grenouille'}
>>> print(type(Dico))
<class 'dict'>
```

Les éléments n'ont pas d'ordre dans un dictionnaire et ne sont donc pas repérés par un indice.

C'est la clé qui permet de repérer une valeur dans le dictionnaire : `dico['Bio1']` renvoie `'poireau'`.

Les clés sont toutes distinctes.

Création d'un dictionnaire	`Dico = {cle1:valeur1, cle2:valeur2, ...}`
Création d'un dictionnaire vide	`Dico_vide = {}`
Ajout d'un élément	`Dico[nvelle_cle] = nvelle_valeur`
Suppression d'un élément	`del Dico(cle)`
Nombre d'éléments dans un dictionnaire	`len(Dico)`

Balayage des clés d'un dictionnaire	`for x in Dico.keys(): # x est une clé`
	`....if Dico[x]=='grenouille':`
	`........print('gagné')`

Test d'appartenance d'une clé à un dictionnaire : `in`	`'PCSI' in Dico.keys()` renvoie `False`

3 Les modules

3.1 Le module math

En général on importe le module `math` sans alias.

```
from math import *
```

Ce module permet d'avoir les fonctions de base : cosinus, sinus, exponentielle, logarithme, ...

pi	3.141592653589793 # ce n'est qu'une valeur arrondie
e	2.718281828459045 # ce n'est qu'une valeur arrondie
racine carré avec `sqrt`	`sqrt(2)` renvoie 1.4142135623730951
logarithme népérien avec `log`	`log(2)` renvoie 0.69314718055994531
logarithme décimal avec `log10`	`log10(100)` renvoie 2.0
Factorielle avec `factorial`	`factorial(10)` renvoie 3628800

3.2 Le module random

En général on importe le module `random` avec l'alias `rd`.

```
import random as rd
```

Ce module permet de réaliser des tirages pseudo-aléatoires de valeurs entières ou réelles.

tirage aléatoire d'un entier entre a et b avec `randint(a,b)` attention le nombre b est inclus	`randint(1,6)` modélise le tirage d'un dé
tirage aléatoire uniforme dans l'intervalle [0, 1[loi de Bernoulli de paramètre p (p dans [0,1])	`random()` renvoie une valeur dans [0, 1[`random()<p` renvoie `True` avec un probabilité p

3.3 Le module numpy

En général on importe le module `numpy` avec l'alias `np`.

```
import numpy as np
```

Le module `numpy` permet l'utilisation d'un nouveau type de variables : les tableaux (type numpy.array).

création avec `np.array` et une liste de sous-listes chaque sous-liste contient les éléments d'une ligne	`T = np.array([[5,0,2],[3,1,0]])` T est un tableau de 2 lignes et 3 colonnes

création d'un tableau de 0
création d'un tableau de 1
	`T = np.zeros( (n,m) )`
	`T = np.ones ( (n,m) )`
	l'argument d'entrée est un tuple (-> doubles parenthèses)

appel d'un élément du tableau extraction d'une partie d'un tableau	`T[0,2]` renvoie l'élément de la ligne 0 et de la colonne 2 `T[2:4,3:]` renvoie une partie du tableau T correspondant aux lignes d'indices 2 et 3 et aux colonnes d'indices 3 jusqu'à la fin
taille du tableau	`(n,m) = np.shape(T)`
concaténation en ligne concaténation verticale	`T = np.hstack((T1,T2))` `T = np.vstack((T1,T2))`

Attention :
- les tableaux peuvent avoir plus de 2 dimensions,
- un tableau n'est pas une matrice : les opérations faites sur un tableau (+, -, *, **, np.cos, np.sin, np.exp, np.sqrt, ...) sont appliquées à chaque élément du tableau.
- `T1 * T2` donne un tableau avec `T[i,j] = T1[i,j] * T2[i,j]` : c'est une multiplication terme à terme.
- pour le produit matriciel il faut utiliser : `P = np.dot(M,N)`.

Utilisation des tableaux pour tracer une courbe :
Pour tracer la courbe représentative de la fonction f sur l'intervalle `[a,b]` il faut :
- créer la liste des abscisses avec `X = np.linspace(a,b,N)` où N est le nombre de points souhaités. Il est également possible d'utiliser `X = np.arange(a,b,pas)` mais cette fois le point b est exclu (comme avec la fonction `range`).
- créer la liste des ordonnées en utilisant la propriété des tableaux (opérations terme à terme). Il suffit d'appliquer la fonction f à X pour obtenir immédiatement Y. Exemple : `Y = 3*np.cos(X) + X**2 - 5` pour la fonction `f(x) = 3cos(x)+x²-5`.
- tracer la courbe avec la fonction `plt.plot` du module `matplotlib.pyplot` : `plt.plot(X,Y)`.
- afficher l'image avec la fonction `plt.show()`.

Utilisation des tableaux pour afficher une image :
Si chaque case d'un tableau correspond à un pixel d'une image il est possible d'afficher l'image correspondante avec `plt.imshow(T)`.
Par défaut la palette de couleur va du violet (associé à la plus petite valeur du tableau) au jaune (associé à la plus haute valeur du tableau).
Il est possible de changer la palette : `plt.imshow(T, cmap='gray')` propose une palette allant du noir au blanc. il existe de nombreuses autres palettes de couleurs.
Il est possible de fixer les valeurs extrêmes pour la palette de couleurs : `plt.imshow(T, cmap='gray', clim = (0,5))` impose le noir pour les valeurs inférieures ou égales à 0 et le blanc pour les valeurs supérieures ou égales à 5. Les valeurs entre 0 et 5 sont des nuances de gris de plus en plus claires.

3.4 Le module copy

En général on importe le module `copy` sans alias.

```
import copy
```

Noter la différence avec l'import du module `math` qui a pour conséquence que l'utilisation d'une fonction du module `copy` doit se faire avec le préfixe `copy.`.
Le module `copy` permet de gérer le phénomène d'**aliasing** qui se produit en Python lorsque que l'on essaie de recopier une liste ou un tableau : l'instruction `L2 = L1` ne donne pas une copie de `L1` mais un **alias** de `L1`, c'est-à-dire un deuxième nom pour désigner la même zone de la mémoire.

Si l'on souhaite faire une copie indépendante d'une liste, il faut utiliser la fonction `copy` du module `copy` : `L2 = copy.copy(L1)`.

Si la liste contient des sous-listes, il faut utiliser la fonction `deepcopy` : `L2 = copy.deepcopy(L1)`.

3.5 Le module pyplot

En général on importe le module pyplot (plus exactement `matplotlib.pyplot`) avec l'alias `plt`.

```
1 import matplotlib.pyplot as plt
```

tracé de fonctions ou affichage d'une image :
cf. le paragraphe correspondant sur le module `numpy`.

options de tracé

`plt.plot(X,Y,'r')`	tracé en rouge
`plt.plot(X,Y,'b')`	tracé en bleu
`plt.plot(X,Y,'g')`	tracé en vert
`plt.plot(X,Y,'w')`	tracé en blanc
`plt.plot(X,Y,'k')`	tracé en noir
`plt.plot(X,Y,'--')`	tracé en pointillés
fermeture de la fenêtre graphique	`plt.close()`
effacement d'une image	`plt.clf()`

Demandez à l'enseignant (ou à Internet) de vous donner les autres options si nécessaire.

4 Les messages d'erreur

NameError : name 'x' is not defined
-> la variable x est utilisée sans avoir été définie. Avez-vous oublié de la mettre en paramètre d'entrée de la fonction ? Est-ce une erreur sur l'orthographe de la variable ?

TypeError : 'str' object does not support item assignment
-> Il est impossible de modifier un caractère dans une chaine de caractères (cf. paragraphe sur les chaines de caractères). `mot[2] = 'b'` n'est pas possible.

IndexError : list assignment index out of range
-> l'indice utilisé est supérieur à la valeur de l'indice maximal (le dernier indice est `len(L)-1`). Regardez bien les limites de la variables servant d'indice...

IndentationError : expected an indented block
-> L'indentation n'est pas respectée. Il y a une tabulation en trop ou en moins.

TypeError : list indices must be integers, not float
-> l'indice utilisé n'est pas un entier. Attention 2.0 n'est pas un entier mais un réel.

TypeError : can only concatenate list (not "int") to list
-> impossible de concaténer une liste L avec le nombre x.
Exemple : `L = L + x` doit être remplacé par `L = L + [x]` ou par `L.append(x)`.

TypeError : f() missing 1 required positional argument : 'y'
-> L'appel à la fonction f ne contenait pas assez de paramètres par rapport à sa définition.
Vérifiez bien la documentation de la fonction utilisée.

TypeError : cannot unpack non-iterable float object
-> La fonction renvoie moins de variables que vous n'en attendez. Exemple : `x,y = f(t)` alors que la fonction ne renvoie qu'une valeur. Python ne peut pas la partager (unpack) entre les deux variables.

SyntaxError : invalid syntax
-> Erreur de syntaxe : une instruction ne se trouve pas au bon endroit. La plupart du temps il manque une parenthèse ou deux points à la fin de la ligne précédente.

TypeError : unsupported operand type for + : 'NoneType' and 'list'
-> `'NoneType'` signifie que la variable n'a jamais reçu de valeur et que Python ne peut donc pas savoir de quel est son type.
L'erreur provient souvent d'une mauvaise utilisation de `append` ou d'une fonction dans laquelle il manque un return.
Par exemple :

```
L = [1, 5] # création de la liste
L = L.append(2) # ajout de 2 à la fin de la liste mais la syntaxe n'est pas correcte : la
    fonction append ne renvoie rien ! L est donc devenu un NoneType object !
L = L + [3] # opération impossible entre un NoneType object et une liste
TypeError: unsupported operand type(s) for +: 'NoneType' and 'list'
```

A vous de compléter avec les erreurs que vous rencontrerez en cours de codage.