GLOSSAIRE

PARTIE 1 : Les premiers mots clé

print() : affiche la variable demandée ou le message proposé entre ' '.
int() : Reconnait comme un entier ou donne la troncature .
float() : Reconnait comme un réel ou convertit en réel si c'est faisable
str() : Reconnait comme une chaine de caractères ou convertit en chaîne de caractères si c'est faisable

input() : attend la saisie d'une donnée par l'utilisateur, instruction à n'utiliser qu'en phase d'apprentissage
Cette fonction peut être combinée afin de demander un entier : int(input()) ou un flottant ; float(input())

PARTIE 2 : Boucles et instructions conditionnelles

<table>
<tr><th colspan="3">CONDITIONS</th></tr>
<tr><td>égalité</td><td>==</td><td>

```
>>> a=15/5  >>> a==3  >>> a==2
              True       False
```

</td></tr>
<tr><td>différent</td><td>!=</td><td>

```
>>> a=15/5  >>> a!= 3  >>> a!=2
              False      True
```

</td></tr>
<tr><td>comparaison</td><td>< ;<= ;> ;>=</td><td>

```
>>> a = 2   >>> a >= 4
              False
```

</td></tr>
<tr><td>et</td><td>and</td><td>

```
>>> a=15  >>> a%5==0 and a%7==0
          False
```

</td></tr>
<tr><td>ou</td><td>or</td><td>

```
>>> a=15  >>> a%5==0 or a%7==0
          True
```

</td></tr>
<tr><th colspan="3">INSTRUCTIONS CONDITIONNELLES</th></tr>
<tr><td rowspan="3">si …….alors …..sinon</td><td>if condition : instruction</td><td>

```
a = int(input("Donner un nombre : "))
if a%7 == 0 : print(a,"est divisible par 7")
```

</td></tr>
<tr><td>if condition : instruction 1
else : instruction 2</td><td>

```
a = int(input("Donner un nombre : "))
if a%7 == 0 : print(a,"est divisible par 7")
else : print(a, "n'est pas divisible par 7")
```

</td></tr>
<tr><td>if condition 1 : instruction 1
elif condition 2 : instruction 2
elif condition 3 : instruction 3
else : instruction 4</td><td>

```
# images par une fonction affine par morceaux
a = int(input("Donner un nombre :"))
if a<=2 : print("L'image de ",a,"est : ",2*a+1)
elif a>2 and a<=5 : print("L'image de ",a,"est : ",a+3)
else : print("L'image de ",a,"est : ",2*a-2)
```

</td></tr>
<tr><th colspan="3">BOUCLE NON BORNÉE</th></tr>
<tr><td>Tant que</td><td>while condition : instruction</td><td>

```
h,n = 90,0 #initialise les valeurs de h et n
while h >= 1:
    h = 0.6*h
    n = n+1
print(n)
```

</td></tr>
<tr><th colspan="3">BOUCLE BORNÉE</th></tr>
<tr><td rowspan="2">pour ….allant de … à …</td><td>for i in range(n) : instruction

range (n) : répète n fois, n prenant ses valeurs entre 0 et n - 1</td><td>

Exemple 1 :

```
for i in range(5) :
    print("Hello")
```

Affichage :

```
Hello
Hello
Hello
Hello
Hello
```

Exemple 2:

```
for i in range(11):
    print("7 * ",i," = ",7*i)
```

Affichage :

```
7 *  0  =  0
7 *  1  =  7
7 *  2  =  14
7 *  3  =  21
7 *  4  =  28
7 *  5  =  35
7 *  6  =  42
7 *  7  =  49
7 *  8  =  56
7 *  9  =  63
7 *  10  =  70
```

</td></tr>
<tr><td>for i in range(a,b) : instruction

range(a,b) où a et b sont des entiers : répète de i = a à i = b - 1</td><td>

Exemple :

```
somme = 0 #initialise somme
for i in range (10,21) :
    somme = somme + i # somme des entiers de 10 à 20
print(somme)
```

Affichage : 165

</td></tr>
</table>

Les boucles ne se finissent pas avec un end, c'est l'indentation qui permet de délimiter les boucles.
Pour une version « papier » des scripts on utilisera le symbole ΕΦ pour signifier une indentation.
Délimiteurs particuliers : les messages de textes sont placés entre « »
les commentaires sont placés après un #

PARTIE 3 : Les Fonctions

GLOSSAIRE

Syntaxe :

```
def {nom de la fonction} ({liste de paramètres})
                                # cette liste peut être vide
    instructions
return {résultat(s)}    # peut renvoyer plusieurs données
```

EXEMPLE 1 : Fonction à 2 paramètres : Calculer le volume d'un cylindre connaissant les dimensions

```
from math import pi
def volume_cylindre(h,R):
    return(pi*R**2*h)
```

Calculer le volume du cylindre de hauteur 5 et de rayon 2,3

Affichage :

```
>>> volume_cylindre(5,2.3)
83.09512568745001
```

EXEMPLE 2 : Fonction avec un paramètre : Somme des n premiers entiers

```
def somme(n):
    S=0
    for k in range(n+1):
        S=S+k
    return(S)
```

Tester pour n = 50.
Tester pour les entiers de 50 à 100

Affichage :

```
>>> somme(50)
1275
>>> somme(100)-somme(49)
3825
```

Une autre technique pour définir les fonctions : **lambda** : cf exercices 29 + document d'accompagnement lycée.

POINT INFO : Les bibliothèques :

Pour l'instant nous utiliserons principalement les bibliothèques **math** et **random**.

Pour avoir la liste des fonctions disponibles dans une bibliothèque on peut utiliser : **import math** puis **dir(math)** :

```
>>> import math

>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', '
atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor
', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma
', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tru
nc']
```

Pour avoir une aide spécifique sur une fonctionnalité, dès que la bibliothèque est importée, on peut utiliser : help().

```
>>> from math import *

>>> help(log)
Help on built-in function log in module math:


log(...)
    log(x[, base])

    Return the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.
```

La bibliothèque **random** :

```
>>>  from random import * # faire appel à toutes les fonctions disponibles dans
cette bibliothèque

>>> randint(1,100) # choisir un entier entre 1 et 100, bornes incluses
78

>>> randrange(2,9,2)  # choisir un entier entre 2 et 9 (9 exclu) en ne prenant
que les nombres de 2 en 2 : 2-4-6-8
2

>>> random() # choisir un nombre (float) entre 0 et 1
0.038025689743349966
```

On prendra l'habitude de n'importer que les fonctionnalités nécessaires au script en construction : comme dans l'exemple 1 où seul le nombre π a été importé :

```
>>> from math import pi
>>> pi
3.141592653589793
```

Lorsque le script fait appel à plusieurs bibliothèques ou lorsqu'il y a un risque de doublon entre une fonction définie par l'utilisateur et une fonction prédéfinie on utilise un alias, par exemple la fonction **randint** du module **random** peut être définie en faisant : import **random** as **rd** puis **rd.randint**.

```
import random as rd
print(rd.randint(1,100))

>>>
75
```

Travail dans la console :

Exemples pour s'approprier les différentes instructions : opérations de base et types de variables.

À saisir	Résultats	Explications
>>> 2 + 3		
>>> "a"+"b"		
>>> 2 * 3		
>>> 2.1 * 5		
>>> "a"*3		
>>> 7 / 3		
>>> 7 // 3		
>>> 15 // 4		
>>> 2 ** 3		
>>> 2 * 3 + 5		
>>> 52 % 15		
>>> 23 % 7		

>>> a = 2
>>> a + 1
>>> "a+1"		

>>> a,b = 2,3 >>> a + b		
>>> 1 > 2		
>>> from math import pi >>> pi < 4		
>>> round(2/3,5)		

C'est dans la console que l'on peut afficher les résultats de fonctions définies dans l'éditeur :

```
def fonction(x) :
    return x**2+5x-9
```

```
def somme_carrés(n):
    S = 0
    for i in range(n):
        S = S + i**2
    return S
```

À saisir	Résultats	Explications
>>> fonction(5)		
>>> fonction(0)		
>>> somme_carrés(12)		
>>> somme_carrés(25) – somme_carrés(12)		

Typage :

Python est un langage à typage dynamique fort, sans précision de l'utilisateur, Python type tout seul.

À saisir	Résultats	Explications
>>> type("abc")		
>>> type(12)		
>>> type(2.0)		
>>> type(2**2+3**2==5**2)		

En revanche il est possible de faire du **transtypage**, à savoir imposer un type à Python, comme par exemple traiter un entier comme une chaîne de caractère.

À saisir	Résultats	Explications
>>> a = str(123)		
>>> b = str(45)		
>>> a + b		
>>> int(a) + int(b)		
>>> int(a+b)		