

Premiers pas avec Python

Chapitre

1

1.1 L'environnement de développement

Un **environnement de développement** ¹ est un ensemble d'outils qui permet d'augmenter la productivité des programmeurs. Un **programme**, ou **script** est une succession de définitions, d'expressions et de commandes. Les **instructions** ²) sont exécutées par l'**interpréteur** Python dans ce que l'on appelle la **console** ³.

¹ en anglais *integrated development environment* (IDE)

² en anglais *statement*

³ en anglais *shell*, contraction de *shell program*








Figure 1.1 – L'EDL en ligne de console.basthon.fr. La console est à droite, l'éditeur de script est à gauche avec une numérotation des lignes. On utilise la touche Exécuter pour lancer le script.

Dans la console, le symbole `>>>` (différent selon les EDL) est un *shell prompt*⁴. Il indique que l'interpréteur est dans l'attente de nouvelles instructions. Vous n'avez pas à le rentrer dans la partie éditeur. Par la suite, la présence `>>>` indique qu'il s'agit d'instructions rentrées directement dans la console.

⁴ to prompt : (of a computer) request input from (a user) “the online form prompts users for data”

1.2 Présentation de la pythonette

- Émulateurs pour smartphone : [Android](#) et [Apple Store](#)
- Simulateurs en ligne [epsilon.nsi.xyz](#) et [numworks.com/fr/simulateur](#)

Dans le menu calcul, découvrir l'utilisation des touches    et  .

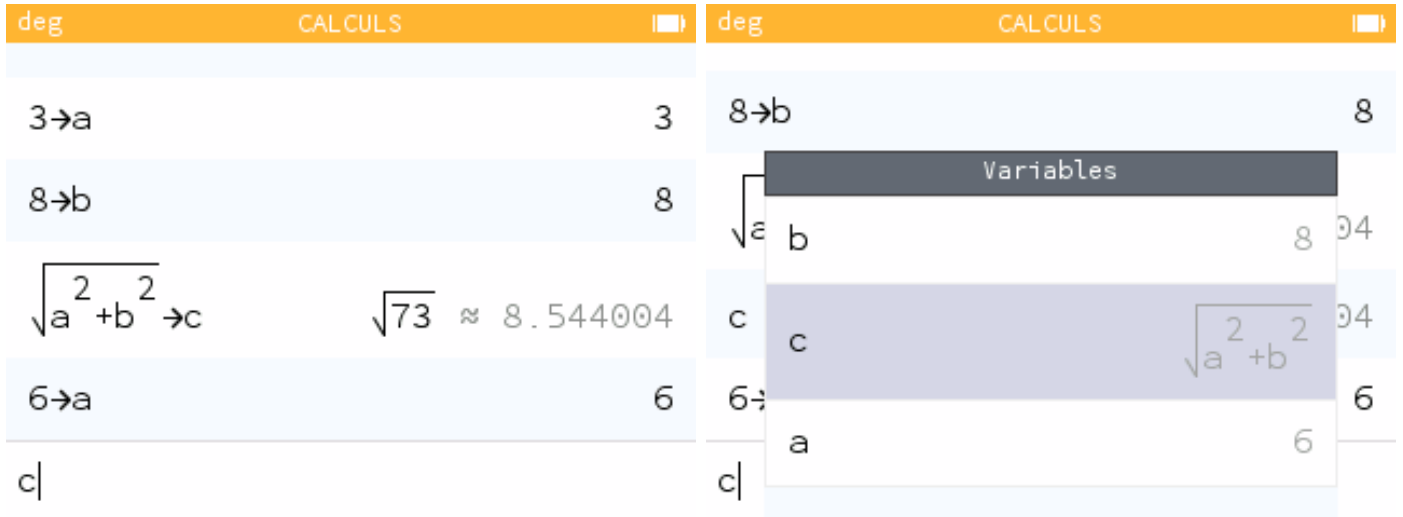










Figure 1.2 – Le menu calcul, expressions et stockage avec la touche .

Dans le menu python, ouvrir un des scripts présent par défaut. Utiliser les touches  et   pour sélectionner un morceau de texte, puis couper/copier/coller/effacer avec les touches    .

Fermer le fichier, et ouvrir une console d'exécution.



Figure 1.3 – Utiliser  puis aller dans > modules > maths

```
>>> 3 * 2
6
>>> pi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pi' is not defined
```

Certaines instructions ne sont pas natives, vous devez les importer :

```
>>> from math import pi # importer l'instruction pi
>>> pi
3.141592653589793
>>> from math import * # tout importer
>>> sqrt(2) # racine carrée
1.4142135623730951
```

Les type de valeurs

```
>>> type(4)
```

```
>>> type(4.0)
```

```
>>> type("4")
```

```
>>> type(3E5)
```

Les opérateurs Dans la console, tester les instructions suivantes et préciser l'opération effectuée :

```
>>> 32 + 10
```

```
>>> 80 - 38
```

```
>>> 6 * 7
```

```
>>> "bon" + "jour"
```

```
>>> 2 * "bon"
```

```
>>> "42" * 3
```

```
>>> 2 ** 6
```

```
>>> 6 ** 2
```

```
>>> 3 ** 2
```

```
>>> 42 / 6
```

```
>>> 42 / 11
```

```
>>> 42 / 0
```

```
>>> 42 // 11
```

```
>>> 42 // 6
```

```
>>> 42 % 11
```

```
>>> 42 % 6
```

Les opérateurs d'affectation Exécuter le script puis remplir le tableau :

```
>>> # à faire dans cet ordre
```

```
>>> a = 101
```

```
>>> b = 4
```

```
>>> e = a * b
```

```
>>> d = e + 2**8 + 6
```

```
>>> a = a - 1
```

```
>>> b = b + 2
```

variable	valeur finale
a	
b	
d	
e	

Exercice 1 — exercices 10 à 13 page 9.**Exercice 2 — <https://bit.ly/2YiTCf3>.**

```

1 a = 5
2 b = 6
3 a , b = b , a

```

1/ Le script affiche :

6, 6	5, 5	5, 6	6, 5
------	------	------	------

```

1 x , y = 1, 10    # double affectation
2 c = y
3 y = x
4 x = c
5 print(x,y)

```

2/ Le script affiche :

10, 10	1, 1	1, 10	10, 1
--------	------	-------	-------

```

1 x=5
2 y=x+14
3 y=y*x
4 y=y+49
5 print(y)

```

3/ Le script affiche :

49	144	54	74
----	-----	----	----

```

1 x = 7
2 y = 2 * x - 1
3 x = x + 3 * y
4 print(x , y)

```

4/ Le script affiche :

130, 13	13, 130	46, 13	13, 46
---------	---------	--------	--------

```

1 a , b = 2 , 3 # double affectation
2 a = a ** b
3 a = a - b
4 b = a - b
5 print(a, b)

```

5/ Le script affiche :

5, 5	3, 3	5, 2	3, 2
------	------	------	------

```

1 a = input("choisir un nombre...")
2 b = 3 * a
3 print(b)

```

6/ On saisit 5. Ce script affiche

15	15.0	555	message erreur
----	------	-----	-------------------

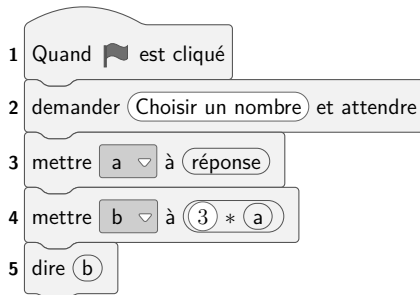
```

1 a = float(input("choisir un nombre..."))
2 b = 2 * a
3 print(b)

```

7/ On saisit 5. Ce script affiche

10	10.0	55	message erreur
----	------	----	-------------------



1.3 Fonctions

Une fonction est un sous programme que l'on peut appeler et exécuter plusieurs fois.

■ **Exemple 1.1 — Un premier exemple.** [lien basthon notebook](#) ou [console](#)

```
1 def mafonctionA ( a ) :
2     b = a**2 - 3 * a + 2
3     return b
4 def mafonctionB ( a ) :
5     b = a**2 + 3 * a + 2
6     print( b )
```

1/ L'appel <code>mafonctionA(1)</code> retourne	0	0.0	<code>None</code>	rien
2/ L'appel <code>mafonctionA(2.0)</code> retourne	0	0.0	<code>None</code>	rien
3/ L'appel <code>mafonctionB(-1.0)</code> retourne	0	0.0	<code>None</code>	rien

```
1 from math import sqrt
2 def hypotenuse(a, b):
3     """théorème de Pythagore"""
4     c = sqrt(a ** 2 + b ** 2)
5     return c
```

4/ L'appel <code>hypotenuse(12,5)</code> retourne	13	13.0	<code>None</code>	42
---	----	------	-------------------	----

```
1 def mafonctionC(b , a) :
2     return 2 * a + b
3 c = mafonctionC(2**3 , 2+3 )
```

5/ L'appel <code>mafonctionC(3 , 2)</code> retourne	8	8.0	7	7.0
6/ L'appel <code>mafonctionC(2 , 3)</code> retourne	8	8.0	7	7.0
7/ La variable <code>c</code> vaut	16	17	18	21

```
1 def salutation() :
2     return "Sire !"
3     return "Bonjour !"
```

7/ L'appel <code>salutation()</code> retourne	<code>"Bonjour !"</code>	<code>"Sire !"</code>	<code>"Sire !" puis "Bonjour !"</code>	<code>"Bonjour !" puis "Sire !"</code>
---	--------------------------	-----------------------	--	--

Exercice 3 — exercices 14 (15, 17), 18, 22 (23) pages 11 à 13.

Exercice 4 lien baston

La fonction suivante calcule le montant de l'impôt d'une personne seule connaissant le montant de son revenu annuel.

```

1 def impotrevenu(r) :
2     """r est le revenu annuel"""
3     if r <= 10084 :
4         impot = 0
5     elif r <= 25710 :
6         impot = 0.11*(r-10084)
7     elif r <= 73516 :
8         impot = 0.30*(r-25710)+impotrevenu(25710)
9     elif r <= 158122 :
10        impot = 0.41*(r-73516)+impotrevenu(73516)
11    else :
12        impot = 0.45*(r-158122)+impotrevenu(158122)
13    return impot

```

a) Quelle valeur retourne `impotrevenu(10084)` ?

b) Quelle valeur retourne `impotrevenu(25710)` ?

c) Quelle valeur retourne `impotrevenu(30000)` ?

d) Rentrer le script sur la pythonette dans un script nommé `p01.py` et vérifiez le fonctionnement à l'aide de l'instruction `impotrevenu(30000)` dans la console. ⁵

⁵ Validation par le professeur □

e) Le taux d'imposition d'une personne est le ratio de l'impôt payé sur le revenu annuel.
Déterminer le taux d'imposition d'une personne seule avec 100000€ de revenu annuel.

1.4 Instructions conditionnelles

■ Exemple 1.2

```

1 chaine = "bateau"
2 if len(chaine) < 6 :
3     print("trop court")
4 print("fini avec la condition")

```

Le test à la ligne 2 vérifie si _____

Comme la condition est _____, le programme affiche _____ puis procède à la ligne 4.

```

1 x = 15926354
2 if x % 6 == 0 :
3     print("multiple de ...")
4 else:
5     print("pas un multiple de ...")
6 print("fin du test")

```

Le test à la ligne 2 vérifie si _____

Comme la condition est _____, le programme va à la ligne _____.

Le programme affiche _____ puis procède à la ligne _____ et affiche _____.

Exercice 5

```

1 a , b = 5 , 9
2 if a>6:
3     b=b-5
4 if b>=13:
5     b=b+9

```

```

1 a , b = 4 , 15
2 if a > 9 :
3     b = b-4
4 if b >= 12:
5     b = b+5

```

1/ Code exécute lignes	2	3	4	5
2/ En fin de script b=	4	9	13	18

```

1 a , b = 10 , 19
2 if a>6:
3     b=b-4
4 if b>=14:
5     b=b+7

```

5/ Code exécute ligne 1	2	3	4	5
6/ En fin de script b=	11	15	16	20

```

1 a , b = 9 , 20
2 if a > 7 :
3     b = b-10
4 if b >= 12:
5     b = b+4

```

3/ Code exécute lignes	2	3	4	5
4/ En fin de script b=	15	19	22	26

7/ Code exécute ligne 1	2	3	4	5
8/ En fin de script b=	10	14	20	24

```

1 a , b = 5 , 18
2 if a > 7 :
3     b = b-10
4 if b >= 12:
5     b = b+4

```

9/ Code exécute lignes

10/ En fin de script b=

2	3	4	5
8	12	18	22

```

1 a , b = 5 , 18
2 if a > 7 :
3     b = b-10
4     if b >= 12:
5         b = b+4

```

11/ Code exécute lignes

12/ En fin de script b=

2	3	4	5
8	12	18	22

```

1 a , b = 5 , 9
2 if a>=0 :
3     b=b-7
4 else :
5     b=b+7
6 if b>0 :
7     a=a+3
8 else :
9     a=a-3

```

13/ Code exécute lignes

14/ En fin de script a=

3	5	7	9
2	5	8	9

```

1 a , b = 7 , 0
2 if a >= 0 :
3     b = b - 7
4 else:
5     b = b + 7
6 if b > 0 :
7     a = a + 9
8 else:
9     a = a - 9

```

17/ Code exécute lignes

18/ En fin de script a=

3	5	7	9
-7	-2	7	16

```

1 a , b = -10 , -15
2 if a >= 0 :
3     b = b - 2
4 else:
5     b = b + 2
6 if b > 0 :
7     a = a + 6
8 else:
9     a = a - 6

```

15/ Code exécute lignes

16/ En fin de script a=

3	5	7
-17	-16	-13

```

1 a , b = 10 , 6
2 if a % 2 == 1 :
3     b = b - 7
4 else:
5     b = b + 7
6 if b % 3 == 2 :
7     a = a + 9
8 else :
9     a = a - 9

```

19/ Code exécute lignes

20/ En fin de script a=

3	5	7	9
1	6	10	19

Exercice 6

```

1 def funcA(arg1) :
2     if arg1 <= 0 :
3         return 0
4     else :
5         return 1

```

```

1 def funcB(arg1) :
2     if arg1 <= 0 :
3         return 0
4     else :
5         return arg1

```

```

1 def funcC(arg1) :
2     if arg1 % 2 == 0 :
3         return arg1//2
4     else :
5         return 3*arg1+1

```

1/ L'appel funcA(3) retourne

2/ L'appel funcA(0) retourne

3/ L'appel funcB(-5.0) retourne

4/ L'appel funcC(12) retourne

5/ L'appel funcC(9) retourne

0	1	1.0	0.0
0	1	1.0	0.0
0	-5	5	-5.0
6	6.0	37	37.0
4.5	27	28	28.0

Exercice 7 — exercices 27, 30, 31, 28 (32) pages 16 et 17.

1.5 Découverte des boucles finies for

`range(fin)` retourne la progression croissante d'entiers consécutifs de 0 inclus à `fin` exclu.

`range(debut, fin)` retourne la progression croissante d'entiers consécutifs de `debut` inclus jusqu'à `fin` exclu.

`range(debut, fin, pas)` retourne la progression croissante d'entiers : `debut`; `debut+pas`; `debut+2*pas`; `debut+3*pas` ... strictement inférieurs à `fin`.

■ **Exemple 1.3** On peut parcourir les éléments d'un objet `range` à l'aide d'une boucle `for`.

```
>>> for i in range(5):
```

```
...     print(i)
```

```
...
```

```
0
```

```
1
```

```
2
```

```
3
```

```
4
```

```
>>> for x in range(3, 8):
```

```
...     print(x)
```

```
...
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
>>> for entier in range(-3,2) :
```

```
...     nbr = entier**2
```

```
...     print(nbr)
```

```
...
```

```
9
```

```
4
```

```
1
```

```
0
```

```
1
```

Exercice 8 — exercices 34, 36 ,39 pages 19 à 23.

Exercice 9

Script A

```
1 n = 10
2 compteur = 0
3 for i in range(1,nbr) :
4     if n % i == 0 :
5         print(i)
6         compteur = compteur + 1
```

Nombre de boucles exécutées =

Script A affiche :

En fin de script A, `compteur`=

Script B

```
1 n = 11
2 compteur = 0
3 for i in range(1,n) :
4     if n % i == 0 :
5         print(i)
6         compteur = compteur + 1
```

Nombre de boucles exécutées =

Script B affiche :

En fin de script B, `compteur`=

Script C

```
1 n = 12
2 compteur = 0
3 for i in range(1,n) :
4     if n % i == 0 :
5         print(i)
6 compteur = compteur + 1
```

Nombre de boucles exécutées =

Script C affiche :

En fin de script C, `compteur`=

■ Exemple 1.4 — Je fais : principe accumulateur pour calculer une somme ou un produit.

```

1 def calculA() :
2     nbr = 0
3     compteur = 0
4     for i in range(100) :
5         nbr = nbr + i
6         compteur = compteur + 1
7     return nbr, compteur

```

```

1 def calculB() :
2     nbr = 0
3     compteur = 0
4     for i in range(5,15) :
5         nbr = nbr + i**2
6         compteur = compteur + 1
7     return nbr, compteur

```

- 1) L'appel `calculA()` exécuteboucles, `calculB()` exécuteboucles
 2) La valeur `nbr` retournée par `calculA()` correspond à la somme :

+ + + + +...+ =

- 3) La valeur `nbr` retournée par `calculB()` correspond à la somme :

+ + + + +...+ =

Exercice 10 — à vous.

```

1 def calculC() :
2     nbr = 0
3     for i in range(5,10) :
4         nbr = nbr + i**3
5     return nbr

```

```

1 def calculD() :
2     nbr = 1
3     for i in range(2,6) :
4         nbr = nbr * i
5     return nbr

```

- 1) L'appel `calculC()` retourne :

+ + + + +...+ =

- 2) L'appel `calculD()` retourne :

```

1 def calcule(n) :
2     nbr = 0
3     for i in range(1, n) :
4         nbr = nbr + 2**(-i)
5     return nbr

```

```

1 def calculF(n) :
2     nbr = 0
3     for i in range(n+1) :
4         nbr = nbr + 4**(-i)
5     return nbr

```

- 3) Les appels `calcule(10)` et `calculF(10)` retournent le résultat des calculs :

Exercice 11 — exercices 43 et 44 page 23.

1.6 Correction et écriture de scripts

Exercice 12 Les scripts ci-dessous doivent retourner respectivement l'aire du rectangle de dimensions `longueur` et `largeur` et l'aire du disque de rayon nommé `rayon`. Corriger les erreurs de chaque fonctions (5 au total).

```
1 def rectangle(longueur, largeur) :
2     return aire
3
4 def fonction (rayon)
5     aire = pi*r*2
6     return aire
```

Exercice 13 La fonction `fonctionB` prend deux arguments nommés `longueur` et `largeur` d'un rectangle, et retourne le périmètre du rectangle correspondant. On souhaite qu'elle retourne `False` si une des dimensions en argument est négative.

Corriger les 6 erreurs de ce script :

```
1 def fonction (longueur; largeur)
2     if longueur < 0
3         return False
4     if largeur < 0
5         return True
6     perimetre=2*longueur+2*largeur
7     return perimetre
```

Exercice 14 Écrit le script d'une fonction d'appel `fonctionC` qui prend un argument nommé `arg` et qui retourne la somme de son carré et de 5.

```
1
2
3
4
5
6
```

Exercice 15 Écrit le script d'une fonction d'appel `fonctionD` qui prend un argument nommé `arg` et qui retourne son carré s'il est positif, et l'opposé du carré s'il est négatif ou nul.

```
1
2
3
4
5
6
7
```

Exercice 16 Écrit le script d'une fonction d'appel `fonctionE` qui prend un argument nommé `arg` et qui retourne sa racine carrée s'il est positif ou nul, et zéro s'il est négatif.

```
1
2
3
4
5
6
7
```

Exercice 17 Écrit le script d'une fonction d'appel `fonctionF` qui prend deux arguments nommés `arg1` et `arg2` et qui retourne `arg2` si `arg1` est positif ou nul, et le triple de `arg2` si `arg1` est négatif strictement.

```
1
2
3
4
5
6
7
```

1.7 Découverte des boucles infinies while

Une boucle infinie `while` est exécutée **tant que** la condition est `True`.

Sa forme générale est :

```
while test == True :
    """une ou plusieurs lignes d'instructions de la
    boucle avec indentation"""
    """sortie de la boucle si test == False"""
```

■ Exemple 1.5

```
1 i = 5
2 while i!=0 :
3     print(i)
4     i = i - 1
5 print(f"sortie i={i}")
```

5 4 3 2 1 sortie i=0

En sortie de boucle

```
1 i = 2
2 while 91 % i !=0 :
3     print(i)
4     i = i + 1
5 print(f"sortie i={i}")
```

2 3 4 5 6 sortie i=7

En sortie de boucle

```
1 i = 10
2 while i**2 < 1664 :
3     i = i + 1
4 print(f"sortie i={i}")
```

sortie i=41

■ Exemple 1.6 En vous aidant du tableau, préciser ce qu'affiche le script suivant :

```
1 n=1
2 while 1.11**n < 2 :
3     n = n+1
4 print(n)
```

n	1	2	3	4	5	6	7	8
1.11**n								

- ☐ Tous les entiers tels que $1.11^n < 2$
- ☐ Le plus grand entier n tel que $1.11^n \geq 2$
- ☐ Le plus petit entier n tel que $1.11^n \geq 2$

- ☐ Rien car il ne s'arrête pas
- ☐ 2.0761602
- ☐ 7

Exercice 18 — à vous. Qu'affichent les programmes suivants. Cochez la bonne réponse.

```
1 n=1
2 while 0.88**n > 0.5 :
3     n = n+1
4 print(n)
```

n	2	3	4	5	6	7	8
0.88**n							

- ☐ 6
- ☐ 5
- ☐ Le plus petit entier n tel que $0.88^n \geq 0.5$

- ☐ Le plus petit entier n tel que $0.88^n \leq 0.5$
- ☐ Rien car il ne s'arrête pas
- ☐ 0.464404

```
1 n=1
2 while 1.15**n < 2 :
3     n = n+1
4 print(n)
```

n	2	3	4	5	6	7	8
1.15**n							

- ☐ 5
- ☐ Le plus petit entier n tel que $1.15^n \geq 2$

- ☐ Rien car il ne s'arrête pas
- ☐ 2,011 357 2