

Annexe A

Premiers pas en Python

A.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.



Figure A.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.

1. en anglais *integrated development environment* (IDE)

2. en anglais *statement*

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

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

A.2 Présentation de la pythonette

A.2.1 Menu calcul

Dans le menu calcul⁵, découvrir l'utilisation des touches    et  .

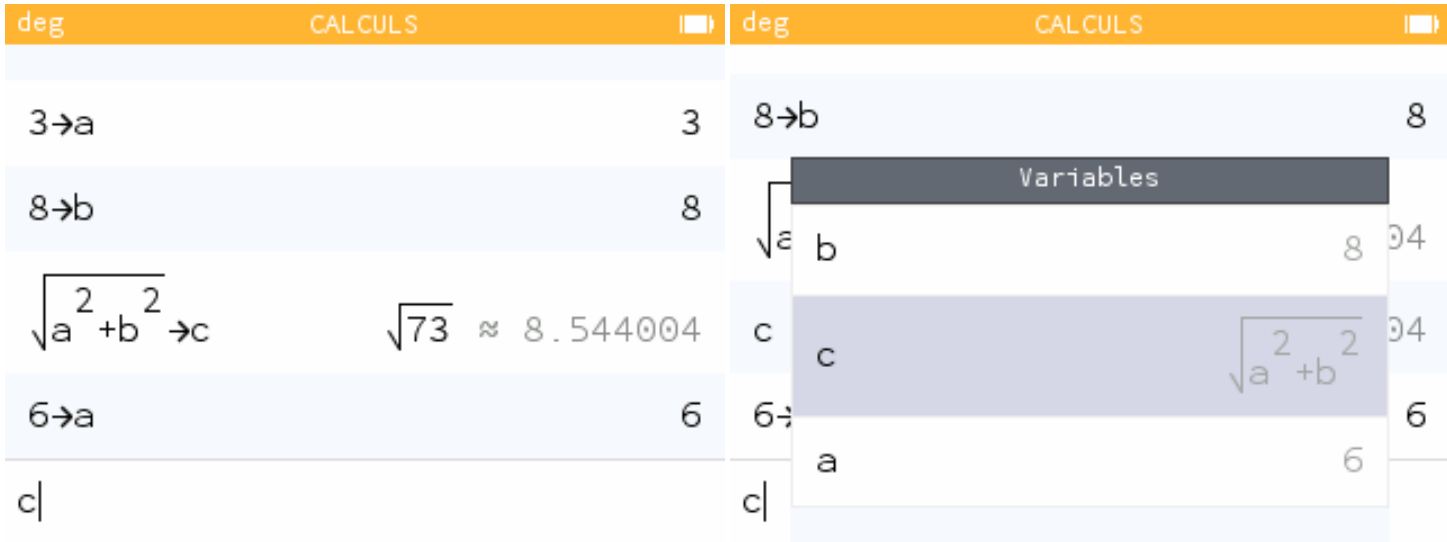











Figure A.2 – Le menu calcul, expressions et stockage avec la touche .

A.2.2 Menu Python

Dans le menu python, ouvrir un des scripts présent par défaut. Vous pouvez 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.

Les types de valeurs

<pre>>>> type(4)</pre>	<pre>>>> type(4.0)</pre>	<pre>>>> type("4")</pre>	<pre>>>> type(3E5)</pre>
<pre><class 'int'></pre>	<pre><class 'float'></pre>	<pre><class 'str'></pre>	<pre><class 'float'></pre>

Les opérateurs mathématiques

à tester	symbole	opération	à tester	symbole	opération
32+10	+		42//11	//	
80-38	-		42//6		
6*7	*		42%11	%	
2**6	**		42%6		
6**2			"pa"+"pi"		
355/113	/		3*"pom"		

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

Les opérateurs d'affectation Une **variable** est un nom que l'on donne à une valeur :

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

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

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

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

variable	a	b	d	e
valeur finale				

```
>>> d = d + 1
```

Les opérateurs de comparaisons


à tester	symbole	compara- raison	à tester	symbole	compara- raison	à tester	symbole	compara- raison
6==3**2			4<4	<		9!=9		
8==2**3	==		4<=4	<=		9!=8	!=	

Quelques instructions natives

<pre>>>> len("Bonjour")</pre>	<pre>>>> min(4**3, 3**4)</pre>	<pre>>>> int("256")</pre>	<pre>>>> chr(42)</pre>
7	64	256	'*'
<pre>>>> max(4**3, 3**4)</pre>	<pre>>>> round(355/113, 2)</pre>	<pre>>>> str(128)</pre>	<pre>>>> ord("*")</pre>
81	3.14	'128'	42

Importer des modules en python

<pre>>>> 3 * 2</pre>	<pre>>>> from math import pi # importer l'instruction pi</pre>
6	<pre>>>> pi</pre>
<pre>>>> pi</pre>	3.141592653589793
Traceback (most recent call last):	<pre>>>> from math import * # tout importer</pre>
File "<stdin>", line 1, in <module>	<pre>>>> sqrt(2) # racine carrée</pre>
NameError: name 'pi' is not defined	1.4142135623730951

Figure A.3 – Certaines instructions ne sont pas natives, vous devez les importer : Utiliser  puis aller dans > modules > maths

A.3 Exercices

A.3.1 Exercices : Calculs, opérations et affectations

Exercice 1 En fin de script :

```
1 A = "pa"
2 B = A + A
3 C = B + "maman"
4 D = A + "radis"
```

A=.....B=.....

C=.....D=.....

Exercice 2 En fin de script :

```
1 x = 5
2 x = x**2
3 x = x-1
4 x = x**2
```

x=.....

Exercice 3 En fin de script :

```
1 x, y = 2, 3
2 z=x+y+x*y
3 y=z/2
```

x=.....y=.....z=.....

Exercice 4 En fin de script :

```
1 x, y = 12, 7
2 u = x+y
3 v=x**2-y**2
4 w=v%u
```

w=.....

Exercice 5 Écrire un script qui :

1. affecte 143 à la variable `a` et 12 à `b`
2. affecte à `c` le reste de la division de `a` par `b`

```
1
2
3
```

Exercice 6 1. Le script affiche :

(A) 6, 6 (B) 5, 5 (C) 5, 6 (D) 6, 5

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

2. Le script affiche (A) 10, 10 (B) 1, 1
(C) 1, 10 (D) 10, 1

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

3. Le script affiche (A) 49 (B) 144 (C) 54
(D) 74

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

4. Le script affiche : (A) 130, 13 (B) 13, 130
(C) 46, 13 (D) 13, 46

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

5. Le script affiche : (A) 5, 5 (B) 3, 3
(C) 5, 2 (D) 3, 2

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

6. On saisit 5. Ce script affiche (A) 15
(B) 15.0 (C) 555 (D) message erreur

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

7. On saisit 5. Ce script affiche (A) 10
(B) 10.0 (C) 55 (D) message erreur

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

A.3.2 Exercices : Fonctions

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

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

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

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

1. L'appel `mafonctionA(1)` retourne
(A) 0 (B) 0.0 (C) `None` (D) rien
2. L'appel `mafonctionA(2.0)` retourne
(A) 0 (B) 0.0 (C) `None` (D) rien
3. L'appel `mafonctionA(-1.0)` retourne
(A) 0 (B) 0.0 (C) `None` (D) rien
4. L'appel `hypotenuse(12,5)` retourne
(A) 13 (B) 13.0 (C) `None` (D) 42
5. L'appel `mafonctionC(3 , 2)` retourne
(A) 8 (B) 8.0 (C) 7 (D) 7.0
6. L'appel `mafonctionC(2 , 3)` retourne
(A) 8 (B) 8.0 (C) 7 (D) 7.0
7. La variable `c` vaut
(A) 16 (B) 17 (C) 18 (D) 21
8. L'appel `salutation()` retourne
(A) "Bonjour !" (B) "Sire !" (C) "Sire !" puis
"Bonjour !" (D) "Bonjour !" puis "Sire !"

Exercice 7 Compléter :

L'appel `f(1)` retourne

L'appel `f(-2)` retourne

L'appel `f(f(0))` retourne

Les arguments de la fonction `affine` sont

L'appel `affine(1,2,3)` retourne

La comparaison `affine(2,1,3)==7` retourne

```
1 def f(x) :  
2     return 3*x-1  
3 def affine(a,b,x) :  
4     return a*x+b
```

Exercice 8 Compléter :

Les arguments de la fonction `volume` sont

L'appel `volume(5,3)` retourne

La comparaison `volume(5,6)>=10` retourne

```
1 def volume(base , hauteur)  
2     v = base*hauteur/3  
3     return v
```

Exercice 9 Écrire une fonction d'appel `g`, d'arguments `x` et `y` et qui retourne le triple de `x` diminué du double de `y`.

```
1 def ..... :
2     return .....
```

A.3.3 Exercices : Instructions conditionnelles

■ Exemple A.2

Script A

```
1 chaine = "bateau"
2 if len(chaine) < 6 :
3     print("trop court")
4 print("suite")
```

Pour le script A, le test à la ligne 2 vérifie si est inférieure à 6.

La valeur de la comparaison est (A) `True` (B) `False`

Le script affiche

Le script va à la ligne 4 et affiche

Script B

```
1 x = 15926354
2 if x % 6 == 0 :
3     print("divisible par 6")
4 else:
5     print("non divisible par 6")
6 print("fin")
```

Pour le script B, le test à la ligne 2 vérifie si par 6.

La valeur de la comparaison est (A) `True` (B) `False`

Le script affiche

Le script va à la ligne

Le script affiche

Script C

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

Pour le script C, la valeur de la comparaison de la ligne 2 est (A) `True` (B) `False`

Après la ligne 3 `a`=..... et `b`=.....

La comparaison ligne 4 est (A) `True` (B) `False`

Le script a exécuté les lignes

Fin de script `a`=.....et `b`=.....

Exercice 10

Script A

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

Script B

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

Script C

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

Script D

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

1/ Script A exécute lignes	2	3	4	5
2/ valeur finale de <code>b</code> =	15	19	22	26
3/ Script B exécute ligne	2	3	4	5
4/ valeur finale de <code>b</code> =	11	15	16	20

5/ Script C exécute lignes	2	3	4	5
6/ valeur finale de <code>b</code> =	10	14	20	24
7/ Script D exécute lignes	2	3	4	5
8/ valeur finale de <code>b</code> =	8	12	18	22

Script E

```

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

```

Script F

```

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

```

Script G

```

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

```

Script H

```

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

```

9/ Script E exécute lignes	3	5	7	9
10/ En fin de script a=	2	5	8	9
11/ Script F exécute lignes	3	5	7	
12/ En fin de script a=	-17	-16	-13	

13/ Script G exécute lignes	3	5	7	9
14/ En fin de script a=	-7	-2	7	16
15/ Script H exécute lignes	3	5	7	9
16/ En fin de script a=	1	6	10	19

Exercice 11

```

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	0	1	1.0	0.0
2/ L'appel funcA(0) retourne	0	1	1.0	0.0
3/ L'appel funcB(-5.0) retourne	0	-5	5	-5.0
4/ L'appel funcC(12) retourne	6	6.0	37	37.0
5/ L'appel funcC(9) retourne	4.5	27	28	28.0

Exercice 12 Compléter :

L'appel location(162) retourne

L'appel location(625) retourne

L'appel location(250) retourne

La fonction location() retourne 75 si

Exercice 13 Compléter :

La fonction a pour appel reponse()

L'..... de la fonction est rep

L'appel reponse("oui") retourne

L'appel reponse("yes") retourne

```

1 def location(x) :
2     if x<=250 :
3         c = 75
4     else :
5         c = 75 + 0.28*(x-250)

```

```

1 def reponse(rep) :
2     if rep == "oui" or rep=="non"
3         return rep
4     else :
5         return "erreur"

```

Exercice 14 Compléter le script de la fonction :

d'appel `fonction()`

d'argument `x`

retourne le double de `x` si `x` est positif

retourne le triple sinon

```
1 def prix(x) :  
2     if 0<x<=20 :  
3         p = 25  
4     elif 20<=x<=50 :  
5         p = 90  
6     elif 50<x<=100 :  
7         p = 150  
8     else :  
9         p = 250  
10    return p
```

Exercice 16 La fonction suivante calcule le montant de l'impôt d'une personne seule connaissant le montant de son revenu annuel.

1. L'appel `impotrevenu(10084)` retourne

2. L'appel `impotrevenu(25710)` retourne

3. L'appel `impotrevenu(30000)` retourne

4. Rentrer le script sur un EDL et vérifier vos réponses

5. 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 def ..... :  
2     if .....  
3         y =  
4     else :  
5         y =  
6     return y
```

Exercice 15 Compléter :

La fonction a pour appel

La fonction a pour argument

L'appel `prix(72)` retourne

L'appel `prix(18)` retourne

L'appel `prix(50)` retourne

L'appel `prix(20)` retourne

L'appel `prix(113)` retourne

L'appel `prix(0)` retourne

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


A.3.4 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 A.3** On peut parcourir les éléments d'un objet `range` à l'aide d'une boucle `for`.

<pre>>>> for i in range(5): ... print(i) ... 0 1 2 3 4</pre>	<pre>>>> for x in range(3, 8): ... print(x) ... 3 4 5 6 7</pre>	<pre>>>> for entier in range(-3,2) : ... nbr = entier**2 ... print(nbr) ... 9 4 1 0 1</pre>
---	--	--

Exercice 17 Choisir la bonne réponse :

- En langage Python lors de instruction `for k in range(2,7)`, `k` prend les valeurs :
(A) 2 et 7 (B) 2; 3; 4; 5 et 6 (C) 2; 3; 4; 5; 6 et 7 (D) 2; 3; 4; 5; 6; 7 et 8
- En langage Python lors de instruction `for k in range(5)`, `k` prend les valeurs :
(A) 0; 1; 2; 3 et 4 (B) 0; 1; 2; 3; 4 et 5 (C) 1; 2; 3 et 4 (D) 1; 2; 3; 4 et 5
- La variable `k` prenne successivement toutes les valeurs entières de 0 à 33 si on utilise
(A) `for k in range(0,33)` (B) `for k in range(0,34)` (C) `for k in range(33)` (D) `for k in range(34)`
- En langage Python, lors de l'instruction `for k in range(2,29)`, `k` prend :
(A) 29 valeurs (B) 28 valeurs (C) 27 valeurs (D) 26 valeurs

Exercice 18 Compléter :

Les valeurs prises par `i` sont

.....

À la fin du script, `a` vaut

À la fin du script, `b` vaut

```
1 a = 0
2 b = 0
3 for i in range(3,17) :
4     a = a +1
5     b = b +1
```

Exercice 19 Compléter :

À la fin du script la variable *a* contient

.....

.....

```
1 a = "Je dis"
2 for i in range(1,5) :
3     a = a+" bravo"
4 a = a+" !"
```

Exercice 20

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 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*=

Script E

```
1 nombre = 0
2 for lettre in "mathematiques" :
3     if lettre == "e"
4         nombre=nombre+1
```

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 D

```
1 p= 10
2 compteur = 0
3 for i in range(5) :
4     p = 2 * p
5     compteur = compteur + 1
```

Nombre de boucles exécutées =

En fin de script D :

compteur=

p=

Nombre de boucles exécutées =

En fin de script E :

nombre=

■ Exemple A.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, et retourne `nbr` correspond à la somme :

+ + + + +...+ =

2. L'appel `calculB()` exécuteboucles et retourne `nbr` correspond à la somme :

+ + + + +...+ =

Exercice 21 — à vous.

```

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

```

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

```

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

```

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 :

A.3.5 Correction et écriture de scripts

Exercice 22 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 23 La fonction `functionB` 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 24 Écrit le script d'une fonction d'appel `functionC` qui prend un argument nommé `arg` et qui retourne la somme de son carré et de 5.

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

Exercice 25 Écrit le script d'une fonction d'appel `functionD` 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 26 Écrit le script d'une fonction d'appel `functionE` 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 27 Écrit le script d'une fonction d'appel `functionF` 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
```

A.3.6 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 A.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

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

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

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

(A) Tous les entiers tels que $1.11^n < 2$ (B) Le plus grand entier n tel que $1.11^n \geq 2$ (C) Le plus petit entier n tel que $1.11^n \geq 2$ (D) Rien car il ne s'arrête pas (E) 2.0761 (F) 7

Exercice 28 — 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							

(A) 6 (B) 5 (C) Le plus petit entier n tel que $0.88^n \geq 0.5$ (D) Le plus petit entier n tel que s'arrête pas (F) 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							

(A) 5 (B) Le plus petit entier n tel que $1.15^n \geq 2$ (C) Rien, il ne s'arrête pas (D) 2,011

A.3.7 TP Aliens

Partie A

validation du professeur

- Scripts 1 à 8 : variables
- Scripts 9 à 20 : boucles for

Partie B

validation du professeur

- Scripts 21 à 26 : conditionnelles
- Scripts 27 à 35 : conditionnelles et boucles for

Partie C

validation du professeur

- Scripts 36 à 47 : boucles infinies while

Partie D

validation du professeur

- Scripts 48 à 56 : fonctions
- Scripts 57 à 62 : boucles for

Partie E

validation du professeur

- Scripts A à F : écrire un script

A.3.8 TP Pyrates

<https://www.py-rates.fr> code de session

Refaire le niveau 1 comme montré puis un maximum parmi les 8 niveaux.

Notation :

- refaire le niveau 1 → 4/20
- faire le niveau 2 → 6/20
- faire le niveau 3 → 8/20
- faire le niveau 4 → 10/20
- faire le niveau 5 → 12/20
- faire le niveau 6 → 14/20
- faire le niveau 7 → 16/20
- faire le niveau 8 → 20/20

A.3.9 Le Module Turtle par l'exemple

Le module `turtle` permet de tracer facilement des dessins en Python. Il s'agit de commander une tortue à l'aide d'instructions simples comme « avancer », « tourner »... C'est le même principe qu'avec Scratch, avec toutefois des différences : tu ne déplaces plus des blocs, mais tu écris les instructions ; et en plus les instructions sont en anglais !

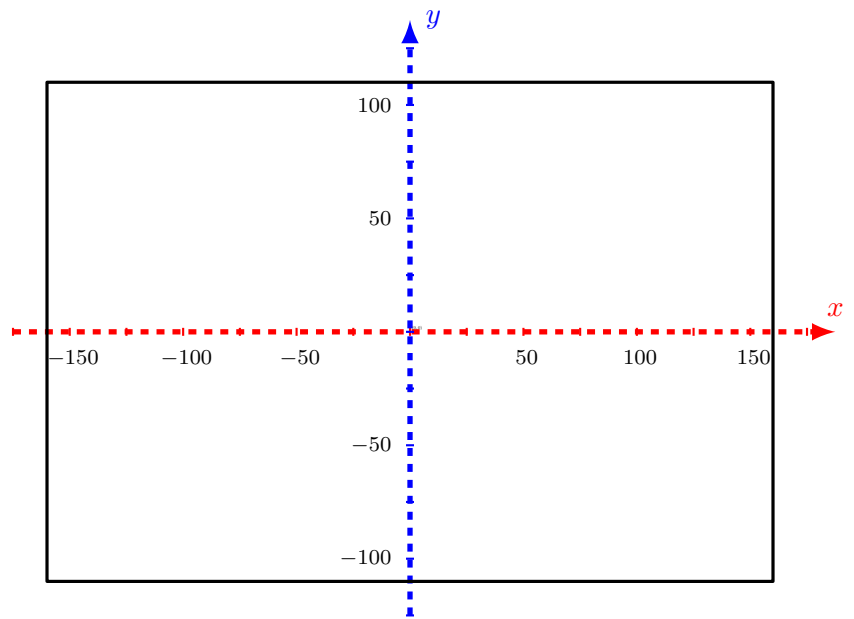


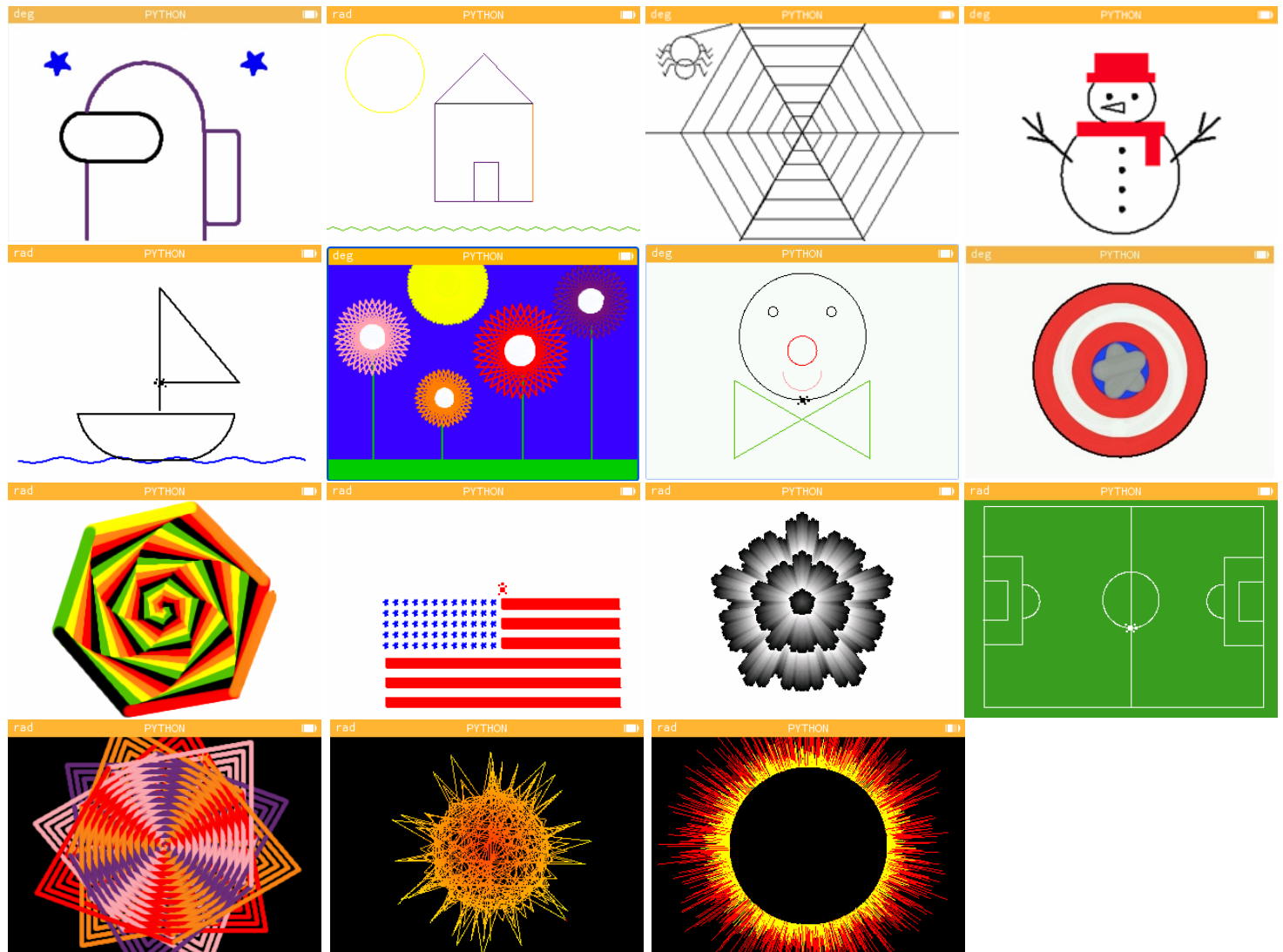
Figure A.4 – Les coordonnées de l'écran par défaut vont de -160 à $+160$ pour les x et de -110 à $+110$ pour les y ; $(0, 0)$ est au centre de l'écran.

- `forward(longueur)` avance d'un certain nombre de pas
- `backward(longueur)` recule
- `right(angle)` tourne vers la droite (sans avancer) selon un angle donné en degrés
- `left(angle)` tourne vers la gauche
- `setheading(direction)` s'oriente dans une direction (0 = droite, 90 = haut, -90 = bas, 180 = gauche)
- `goto(x,y)` se déplace jusqu'au point (x, y)
- `setx(newx)` change la valeur de l'abscisse
- `sety(newy)` change la valeur de l'ordonnée
- `down()` abaisse le stylo
- `up()` relève le stylo
- `width(epaisseur)` change l'épaisseur du trait
- `color(couleur)` change la couleur : "red", "green", "blue", "orange", "purple",...
- `position()` renvoie la position (x, y) de la tortue
- `heading()` renvoie la direction `angle` vers laquelle pointe la tortue
- `showturtle()` et `hideturtle()`

A.3.10 DM : Les mathématiques sont belles !

Réaliser avec votre calculatrice une belle construction soit en écrivant un algorithme python qui exploite le module turtle ou le module kandinsky.

Quelques exemples de ce qu'il est possible de faire avec Turtle



source : Vincent Roche

Consignes :

1. Le travail doit être réalisé sur votre calculatrice, la NumWorks. Si vous n'avez pas cette calculatrice, vous pourrez utiliser l'émulateur en ligne, il est gratuit.
2. N'hésitez pas à consulter des exemples de réalisations, vous verrez ainsi ce qu'il est possible de réaliser. Copier un travail déjà fait est une très mauvaise idée.
3. Vous pouvez également faire des recherches sur internet, mais là encore copier un script sans faire aucune modification est risqué.

4. Vos meilleures réalisations seront soumises à NumWorks. Des coques personnalisée pour leur calculatrice sont à gagner pour des concours régulier.
 5. Vous devez conserver une trace de vos recherches sur internet et des ressources que vous avez exploitées, elles vous seront demandées dans le compte rendu électronique. (Sur un ordinateur PC, appuyez sur CTRL + H pour récupérer l'historique de votre navigateur web et consulter cette page pour retrouver l'historique associé à votre compte google.)
 6. Vous pouvez faire les essais sur votre calculatrice, et avancer ce travail pendant vos heures d'études par exemple, mais il sera impératif de le finaliser sur l'espace « Mes scripts » depuis <https://my.numworks.com/python/> . Il faut ici aussi de connecter avec son compte NumWorks puis se rendre dans « mes scripts »:
 - Laissez bien la case « Permettre aux autres utilisateurs d'accéder à ce script »coché sinon il sera impossible de voir le travail rendu.
 - Clic droit sur l'écran de la calculatrice et enregistrer l'image sous
 7. L'image et le lien hypertexte devront être rendus via un formulaire électronique
- Par exemple, le lien https://my.numworks.com/python/niz-moussatat/turtle_prof_001 donne le

```
1  # Premier exemple
2  from turtle import *
3  penup()
4  pas = 40
5  goto(0,0)
6  setheading(0)
7  pendown()
8  width(3)
9  for i in range(25) :
10     color(255-10*i,10*i,10*i)
11     forward(pas)
12     left(90)
13     forward( 2*pas )
14     goto(0,0)
15     pas = pas -1
16  penup()
```

