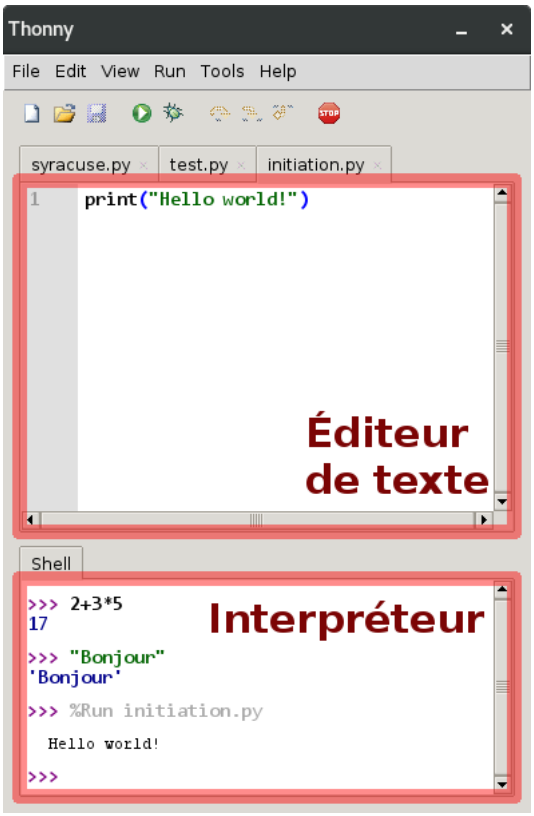
Prise en main de Python

Thonny

Python est un langage **interprété**. C'est-à-dire qu'on peut écrire un programme entier et ensuite l'exécuter ou simplement taper les commandes une par une dans un **interpréteur** et voir les résultats au fur et à mesure.

Pour travailler avec Python, nous utiliserons Thonny qui est un "IDE" (integrated development environment) qui permet à la fois d'écrire des programmes dans l'éditeur de texte (fenêtre du haut), de les exécuter et aussi d'interagir avec l'interpréteur (fenêtre du bas).
L'interpréteur agit comme une calculatrice. Il donne le résultat de chaque opération qu'on lui donne.




```
>>> 2 + 3 * 5
17
```

Si on donne un texte à l'interpréteur, il l'affiche en réponse :

```
>>> "Bonjour"
'Bonjour'
```

On peut aussi écrire un programme dans l'éditeur de texte :

```
print("Hello world!")
```

Pour l'exécuter, il faut cliquer sur le bouton "Run" (). Si le fichier n'a pas été sauvegardé, Thonny va vous demander de l'enregistrer. Vous pouvez l'appeler `initiation.py`. Le programme s'exécute et affiche le résultat dans l'interpréteur.

```
>>> %Run initiation.py
Hello world!
```

À partir de là, à chaque fois que vous ferez une modification dans votre programme, il faudra à nouveau l'exécuter pour voir le résultat. Lorsque vous appuyez sur "Run" (ou sur F5), le programme est sauvegardé et exécuté.

À partir de maintenant la couleur de fond des cadres indiquera s'il faut mettre le texte dans l'éditeur ou dans l'interpréteur :

```
# Si le fond est bleu, il faut écrire les commandes dans l'éditeur de texte.
```

```
# Si le fond est rouge, il faut écrire les commandes dans l'interpréteur.
# Il faut juste copier les commandes se trouvant après >>>
>>> COMMANDE A TAPER
Resultat obtenu, à ne pas recopier
```

Arithmétique et variables

Dans cette partie, nous utiliserons uniquement **l'interpréteur** (la fenêtre du bas).
Comme nous l'avons vu, l'interpréteur peut être utilisé comme une calculatrice.
Les opérations de base en Python sont données dans le tableau ci-contre.
Par exemple $\dfrac{5\times(7-4)}{2}$ s'écrit `(5 * (7-4)) / 2`

En math	En Python
$x+y$	`x + y`
$x-y$	`x - y`
$x\times y$	`x * y`
$x:y$	`x / y`
x^2	`x**2`
x^3	`x**3`
x^y	`x**y`

```
>>> (5 * (7-4)) / 2
7.5
>>> (3 + 4 + 5 + 6)**2
324
```

Remarque :
En Python, les nombres décimaux s'écrivent avec un "." à la place de la virgule. Dans l'exemple précédent le premier résultat est décimal.
Python distingue les nombres entiers et décimaux. Ainsi, `2.0` est un nombre décimal, alors que `2` est entier.

Exercice 1 :
Traduire en Python les expressions suivantes et déterminer leur résultat :

a) 3^2+2^3 b) $\dfrac{5}{3}\times\dfrac{9}{2}+12$ c) $\left(\dfrac{36}{25}\right)^2$

Pour la division euclidienne, il faut utiliser `x // y` pour le quotient et `x % y` pour le reste.
Le reste de la division euclidienne, aussi appelé **modulo**, est très utilisé en Python.

```
>>> 17 / 5      # Division sur les réels
3.4
>>> 17 // 5     # Quotient de la division euclidienne
3
>>> 17 % 5      # Reste de la division euclidienne
2
```

On peut utiliser des **variables** pour stocker le résultat de calculs.

Définition :
En Python, le nom d'une **variable** doit respecter les règles suivantes :
- commencer par une lettre ou un "_" ;
- contenir des lettres, des chiffres et des "_".

Pour **déclarer** une variable, on doit lui donner une valeur par une **affectation** :

`variable = expression`

```
>>> une_variable = 0
>>> UneAutreVariable = "Bonjour"
>>> _une_3eme = 7
>>> _4_fois_3 = 12
```

Pour obtenir la valeur d'une variable il faut juste taper son nom dans l'interpréteur ou utiliser la commande **print**(variable).

```
>>> a = 1
>>> a
1
>>> print(a)
1
```

Une fois la variable affectée, on peut l'utiliser dans des opérations.

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

Il est aussi possible de modifier la valeur d'une variable.

```
>>> a = 5
>>> a
5
>>> b
2
```

On remarque que la valeur de b n'a pas été modifiée. Une fois qu'une variable est affectée, sa valeur ne dépend plus des variables utilisées dans l'expression de l'affectation. On peut aussi afficher les valeurs de plusieurs valeurs à la fois.

```
>>> a, b
(5, 2)
>>> print(a, b)
5 2
```

De façon générale, **print** peut afficher des valeurs et des textes, en les séparant par des virgules. Chacune des valeurs sont séparés par des espaces dans le message obtenu.

```
>>> print("3 *", a, "=", 3*a)
3 * 5 = 15
```

On peut utiliser la valeur actuelle d'une variable pour lui en donner une nouvelle.

```
>>> a = a + 1
>>> a
6
>>> a = a + 1
>>> a
7
```

Remarque :
Pour ajouter une valeur à une variable, il est possible d'utiliser la commande :

```
variable += expression
```

Cela revient à faire :

```
variable = variable + expression
```

De la même manière il est possible d'utiliser :

```
variable -= expression
variable *= expression
variable /= expression
variable //= expression
variable %= expression
```

Exercice 2 :

Quelle est la valeur de `secret` à la fin de cette séquence d'instructions ?

```
public = 2695
secret = public**2
secret //= 135
secret -= 65
secret %= 53
secret += 126
```

Les noms en Python sont sensibles à la casse. Cela veut dire que Python fait la différence entre majuscule et minuscule. Ainsi `nom`, `NOM`, `Nom` ou `nOm` sont considérés comme des noms différents.

```
>>> nom = "Python"
>>> Nom
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Nom' is not defined
```

Remarque :
Lorsque vous avez une erreur du type :

```
NameError: name 'VAR' is not defined
```

Cela veut dire que la variable `VAR` n'a pas encore été utilisée et n'a pas de valeur. Il faut soit lui donner une valeur en amont, soit vérifier que l'orthographe du nom de la variable est correct.

Fonctions en Python

Dans Python, il est possible de définir des fonctions. Elles permettent de rajouter de nouvelles commandes à Python.
Pour définir une fonction, il faut utiliser la syntaxe :

```
def nom_de_la_fonction(parametres):
    instruction_1
    instruction_2
    ...
    intruction_n
```

Il faut commencer par utiliser le mot-clef **def** suivi du nom de la fonction. Ensuite il faut mettre, entre parenthèses, les paramètres de la fonction, séparés par des virgules. S'il n'y a pas besoin de paramètres, comme dans notre exemple, il faut quand même mettre des parenthèses vides.
La première ligne doit se terminer par ":". Cela indique à Python que la suite correspond aux instructions de la fonction. Chacune des instructions de la fonction doit être indentée de la même manière. C'est-à-dire qu'il doit y avoir autant d'espaces devant chacune des lignes. Si les lignes ne sont pas indentées correctement, vous aurez le message d'erreur suivant :

```
>>> def test():
        x = 1
         y = 2

  File "<pyshell>", line 3
    y = 2
    ^
SyntaxError: unexpected indent
```

Cela signifie que l'indentation de la commande `y = 2` ne correspond pas à ce qui précède.
Si on souhaite rajouter des instructions après la définition d'une fonction, il suffit de revenir au début de la ligne.

```
def bonjour():
    nom = input("Quel est ton nom ? ")
    print("Bonjour", nom)
```

Pour appeler une fonction, il suffit de taper son nom, suivi des valeurs des paramètres entre parenthèses. Même s'il n'y a pas de paramètre, il faut mettre les parenthèses.

```
>>> bonjour()
Quel est ton nom ? Bob
Bonjour Bob
```

Par contre, la variable `nom` est une "variable locale" de la fonction. Sa valeur n'est pas accessible après l'appel à la fonction.

```
>>> nom
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
NameError: name 'nom' is not defined
```

Pour pouvoir garder la valeur rentrée par l'utilisateur après l'appel à la fonction, il faut rajouter une ligne à la fonction :

```
def bonjour():
    nom = input("Quel est ton nom ? ")
    print("Bonjour", nom)
    return nom
```

La fonction se comporte alors comme une fonction en mathématiques. C'est-à-dire que l'appel à la fonction produit une valeur :

```
>>> nom = bonjour()
Quel est ton nom ? Bob
Bonjour Bob
>>> nom
'Bob'
```

La commande **return** permet de garder la valeur d'une variable ou d'une expression à la fin de l'exécution d'une fonction.
On peut utiliser un autre nom de variable :

```
>>> camion = bonjour()
Quel est ton nom ? Semi-remorque
Bonjour Semi-remorque
>>> camion
'Semi-remorque'
>>> nom
'Bob'
```

La variable nom a gardé sa valeur.
Si on veut passer des valeurs à la fonction, il suffit de rajouter des paramètres dans les parenthèses lors de la définition.

```
def bonjour(nom):
    print("Bonjour", nom)
```

Pour appeler la fonction, il faut maintenant mettre entre parenthèse une valeur pour le paramètre. On dit que cette valeur est un **argument**.

```
>>> bonjour("Bob")
Bonjour Bob
```

Le paramètre de la fonction est nom et l'argument est "Bob". Lors de l'appel de la fonction, on affecte la valeur de l'argument au paramètre.
S'il y a plusieurs paramètres, il faut les séparer par des virgules.

```
def bonjour2(nom1, nom2):
  print("Bonjour", nom1, "et", nom2)
```

```
>>> bonjour2("Alice", "Bob")
Bonjour Alice et Bob
```

Il est aussi possible de définir des valeurs par défaut. Pour cela, on affecte directement une valeur au ou aux paramètres concernés. Ces valeurs ne sont utilisées que si aucun argument n'est donné pour ce paramètre.

```
def bonjour2(nom1, nom2="les autres"):
  print("Bonjour", nom1, "et", nom2)
```

```
>>> bonjour2("Alice", "Bob")
Bonjour Alice et Bob
>>> bonjour2("Alice")    # on ne donne pas de valeur pour nom2
Bonjour Alice et les autres
```

Fonctions et calculs de valeurs

On peut définir des fonctions qui calculent un résultat.

```
def f(x):
    n = x - 2
    n = 3 * n * n
    n = n - 5
    return n
```

Cette fonction effectue un programme de calcul et correspond à une fonction “classique” en mathématiques.

```
>>> f(0)
7
>>> a = f(1)
>>> a
-2
```

On peut soit l’utiliser sans affecter sa valeur, et elle sera affichée dans l’interpréteur, ou on peut l’affecter à une variable pour pouvoir l’utiliser plus tard.
On peut également l’utiliser dans des calculs ou l’appeler avec des expressions plus complexes :

```
>>> f(2) + f(3)
-7
>>> f(1+5)
43
>>> f(f(4))
70
```

Il est possible de calculer directement le résultat en mettant une expression dans le **return** :

```
def g(x):
    return 3*x*x - 12*x + 7
```

```
>>> g(1)
-2
>>> g(f(1))
43
```

Si vous voulez utiliser le résultat de votre fonction pour l’affecter à une variable ou pour un calcul, il faut absolument utiliser **return**. Si la fonction doit “renvoyer” une valeur, il faut utiliser un **return**. Sinon, un **print** peut suffire. La plupart des fonctions que nous programmerons utiliseront un ou des **return**.

Exercice 3 :

Recopier et compléter la fonction `somme_carres(x, y)` qui renvoie la somme des carrés de `x` et de `y`.

```
def somme_carres(x, y):
    x_carre = x**2
    y_carre = ...
    return ... + ...
```

```
>>> somme_carres(3, 4)
25
>>> somme_carres(11, 58)
3485
```

Exercice 4 :

Recopier et compléter la fonction `carre_somme(x, y)` qui renvoie le carré de `x + y`.

```
def carre_somme(x, y):
    somme = ...
    carre = ...
    return ...
```

```
>>> carre_somme(3, 4)
49
>>> carre_somme(11, 58)
4761
```

Exercice 5 :

Recopier et compléter la fonction `seconde_vers_heures(secondes)` qui prend une durée en secondes et affiche le nombre d'heures, de minutes et de secondes qui correspondent.

```
def seconde_vers_heures(secondes):
    minutes = secondes // 60
    secondes = secondes % 60
    heures = ... // 60
    minutes = ... % 60
    print(heures, "h", minutes, "m", secondes, "s")
```

```
>>> seconde_vers_heures(79)
0 h 1 m 19 s
>>> seconde_vers_heures(9837)
2 h 43 m 57 s
```

Puisque cette fonction utilise un **print** et pas un **return**, il n'est pas possible d'utiliser le résultat de la fonction pour faire un calcul, ni de le stocker dans une variable.

Exercice 6 :

Recopier et compléter la fonction `imc(masse, taille)` qui prend une masse en kg et une taille en m et qui renvoie l'indice de masse corporelle selon la formule $\frac{\texttt{masse}}{\texttt{taille}^2}$.

```
def imc(masse, taille):
    return ...
```

```
>>> imc(80, 1.75)
26.122448979591837
>>> imc(60, 1.60)
23.437499999999996
>>> imc(80, 1.60)
31.249999999999993
```

Exercice 7 :

Recopier et compléter la fonction `moyenne_trois_notes(note1, note2, note3)` qui prend trois notes sur 20 et renvoie la note moyenne.

```
def moyenne_trois_notes(note1, note2, note3):
    somme = ...
    moyenne = ...
    return ...
```

```
>>> moyenne_trois_notes(0, 10, 20)
10.0
>>> moyenne_trois_notes(12, 18, 15.5)
15.166666666666666
>>> moyenne_trois_notes(12, 18, 15)
15.0
>>> moyenne_trois_notes(12, 18, 12)
14.0
```

Exercice 8 :

Rajoutez les fonctions `fonction1(x)` et `fonction2(x)` correspondant aux programmes de calcul ci-dessous, où `x` correspond au nombre choisi au début :

Programme 1 :
- Choisir un nombre ;
- Le mettre au carré ;
- Ajouter 9.

Programme 2 :
- Choisir un nombre ;
- Soustraire 3 ;
- Calculer le carré du résultat ;
- Ajouter 6 fois le nombre de départ.

Fonctions et erreurs

Lors de l'évaluation d'une fonction, le calcul s'arrête dès qu'une commande **return** est lue, même s'il reste des instructions dans le corps de la fonction.

```
def h(x):
    return 1
    return 2
```

La fonction précédente va toujours renvoyer la valeur 1 :

```
>>> h(0)
1
>>> h(1)
1
```

Il faut donc faire attention, lorsqu'il y a plusieurs **return** dans une fonction que ce soit le bon qui soit utilisé dans chacun des cas.
Il peut aussi arriver qu'on oublie de mettre un **return**, alors qu'on en a besoin.

```
def oups(x):
    x = x + 1
```

```
>>> oups(1)
>>> a = oups(2)
>>> a    # on remarque qu'il n'y a pas de valeur affichée
>>> a + 1
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

On pourrait croire que `oups(1)` ou `a` n'ont pas de valeur. En fait ils ont la valeur **None**. Cette valeur spéciale correspond à une valeur nulle. Si une variable est associée à cette valeur nulle, il n'y a rien qui s'affiche si on tape juste le nom de la variable dans l'interpréteur. S'il n'y a pas de **return** rencontré lors de l'évaluation de la fonction, le résultat renvoyé sera toujours **None**.
Par contre, si on utilise la valeur **None** dans un calcul, on obtient le message d'erreur ci-dessus. Si vous voyez ce message, c'est probablement parce que vous avez oublié un **return** quelque part ou que vous utilisez une fonction qui ne renvoie pas de valeur.