Python est un langage de programmation **interprété** créé par *Guido Van rossum* en 1991. Contrairement à **Scratch**, vous devez taper le code que vous voulez exécuter. Il y a donc des instructions en anglais à connaitre : on appelle cela la **syntaxe**. Ce code se tape dans un environnement de travail où il sera interprété. Vos programmes seront enregistrés sous le nom *Turing*, appuyez sur F5 pour les interpréter.

Python est un logiciel libre crée en 1991.

0.1 Les variables

Une variable est l'association entre un nom et une valeur. L'affectation est une instruction qui permet de donner une valeur à une variable. Par exemple, vous écrivez $A \longleftarrow 5$ en pseudo-code (type BAC). Python utilise le symbole = pour cela : `A = 5`. Cela n'a donc rien à voir avec l'égalité au sens mathématique.

Exemple 1 Sans taper le code, indiquer les valeurs de a, b et c après exécution de la séquence ci-contre :

```
a = 3
b = 6
c = 1
a = b
b = a
c = b
```

Exemple 2 Implémentez l'algorithme suivant :

- Choisir un nombre.
- Le multiplier par 2
- Ajouter 5.
- Multiplier le résultat par le nombre de départ.
- Afficher le résultat.

Calculez le résultat pour $x = -1$, $x = 10$ et vérifiez que votre programme affiche le bon résultat. Vous pouvez faire entrer une valeur à l'utilisateur avec la fonction `input()`:

```
x = int(input("Entrez une valeur pour x"))
```

mais il est préférable de simplement affecter la valeur à la main dans votre code.

En bref Une variable possède un nom ou *identificateur* qui fait référence à un emplacement de la mémoire de l'ordinateur contenant un objet. Toute variable est *in fine* représentée pour le processeur par des octets, et c'est grâce au type de cette variable que le processeur saura si ces octets représentent un entier, un réel, une chaîne de caractères etc. Python est un langage typé **dynamiquement**, le type des variables est déterminé lors de l'interprétation et pourra éventuellement être changé. A contrario, un langage typé statiquement comme C/C++ ou Java force à définir le type des variables et à le conserver au cours de la vie de la variable.

Cette facilité de Python peut également être source de confusion et il convient d'avoir bien réfléchi au type des données que l'on utilise. Nous y reviendrons au moyen des annotations.

Syntaxe dans différents langages

En **C et C++**	En **Python**	En **Javascript**
int a = 2	a = 2	**var** a = 2
float b = 3.14	b = 3.14	**var** b = 3.14

- Le symbole `=` permet d'effectuer une *affectation* et non un test d'égalité.
- En C/C++, le type de la variable a du être déclarée: b est de type `float` (nombre à virgule) et a de type `int` (nombre entier). Ce type ne pourra plus être changé.
- En Python et Javascript, le type des variables est géré par le langage et pourra éventuellement être changé. En Javascript, le mot clé `var` permet de déclarer une variable globale, `let` permet de déclarer une variable dont la portée est limitée à un bloc : on dit qu'elle est **locale** à ce bloc.

On dit que C est un langage à **typage statique**, alors que Python et Javascript sont à **typage dynamique**.

0.2 Activité: structures algorithmiques

Dessiner avec le module Turtle de Python Le module *turtle* est un outil du langage *Python* de tracé de figures simples, il a l'avantage de pouvoir être utilisé avec très peu de connaissances. Un curseur (la tortue) effectue le tracé à l'écran en se déplaçant selon les instructions codées par l'utilisateur. L'entête de votre code devra comporter l'importation de ce module:tapez la ligne *from turtle import **.

- `forward(N)` avance le curseur de N pixels.
- `backward(N)` recule le curseur de N pixels.
- `left(`α`)` tourne le curseur de $\alpha°$ vers la gauche.
- `goto(x,y)` déplace le curseur au point de coordonnées *(x,y)*.
- `up()` monte le crayon: le tracé ne s'effectue plus.
- `down()` descend le crayon.
- `color`(couleur) colorie le tracé.
- `begin_fill()` active le mode remplissage.
- `end_fill()` désactive le mode remplissage.
- `fillcolor`(couleur) sélectionne la couleur de remplissage.

L'adresse suivante vous fournira l'ensemble des instructions disponibles:

`http://fr.wikibooks.org/wiki/Programmation_Python/Turtle`

0.2.1 Séquence

A faire:

1. Dessiner un carré, puis deux carrés adjacents.
2. Dessiner un hexagone puis un décagone.

En bref La séquence est une suite d'instruction. Dans ce bloc, les instructions sont exécutées les unes après les autres. Les instructions peuvent contenir des expressions. Une expression est une suite de caractères définissant une valeur. Pour connaître cette valeur, la machine doit évaluer l'expression.

Exemple: `10//3` est une expression, elle est évaluée à 3.

0.2.2 Boucle

Un ordinateur est plutôt doué pour le travail répétitif: les *boucles* sont faites pour répéter les séquences d'instruction. Lorsqu'on connaît le nombre de répétition à effectuer, la boucle est dite bornée. On utilise une boucle `pour`.

CC-BY-SA

Exemple: Python

```
for i in range(3):
    print(i)
print('Vive la NSI')
```

```
>>> %Run Exemple.py
0
1
2
Vive la NSI
```

Le compteur de boucle `i` est un entier, par défaut le pas est de 1.

A faire:

1. Comment modifier le code pour afficher trois fois 'Vive la NSI' ?
2. Reprenez vos réponses aux questions 2.1.1 et 2.1.2 avec une boucle *pour*.

En bref Les syntaxes en **C** et **Javascript** sont identiques pour la boucle **Pour**. Les blocs d'instructions sont placés entre accolades en C et Javascript. En Python, on utilise deux points : suivis d'un décalage du code appelé **indentation**. Cette indentation est obligatoire en Python pour marquer le bloc à répéter. Elle est conseillée dans les autres langages.

Syntaxe dans différents langages

En **C**	En **Python**	En **Javascript**

for (i=0; i < 11; i++)
{
printf(*i*);
}
	for i **in range**(0, 10):
	print(i**2)
		for (i=0; i < 11; i++)
		{
		document.write(i*i);
		}

0.2.3 Les fonctions

On peut définir dans tous les langages de programmation des fonctions, celles-ci peuvent être appelées à chaque fois qu'une même séquence d'instruction est nécessaire. En voici la syntaxe Python, les lignes précédées d'un # sont des commentaires:

```
#Définition de la fonction
def hello_world():
    print("Bonjour à tous !")
#Appel de la fonction
hello_world()
```

Le mot clé `def` sert à définir la fonction dont le nom est `hello_world`. Celle-ci est appelée pour être exécutée par `hello_world()`.

: Il est essentiel de distinguer le début et la fin du bloc d'instructions:

- `print("Bonjour à tous !")` est une instruction incluse dans le **corps de la fonction**.
- L'appel de la fonction `hello_world()`ne l'est pas: cette instruction est en dehors du **corps de la fonction**.

Le décalage du code appelé **indentation** permet en Python de distinguer les limites de bloc.

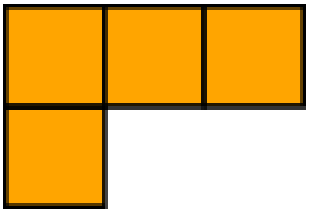


Figure 1: Tétromino L ou lambda

A faire:

1. Reprendre le code de votre carré et écrire une fonction `carre` qui dessinera votre carré dès qu'elle sera appelée.

2. Le jeu Tétris créé en 1984 doit son nom aux pièces à assembler au cours du jeu constituées de quatre carrés (en grec ancien, *tetra* signifie *quatre*). Ces pièces sont appelées tétrominos. En utilisant la fonction `carre`, écrire une fonction `tetro_L` dessinant le tétromino nommé *lambda* représenté sur la figure 1.

3. Ecrivez une fonction `hexagone` et `decagone`.

4. Nous aimerions dessiner des polygones de mesure différente à chaque appel, il nous faudra pour cela utiliser un **paramètre**. Celui-ci se comporte comme une variable locale à la fonction et sa valeur sera donnée à chaque appel par l'utilisateur. Une fonction peut prendre un ou plusieurs paramètres que l'on écrit entre les parenthèses. On passe les valeurs (appelées arguments) au moment de l'appel de la fonction, en voici deux exemples: Un premier en *Python* où on appelle la fonction avec l'instruction `double()` dans le shell et un second en *Javascript* :

```
def double(x):
    resultat = 2*x
    return resultat


>>>double(5)
10
```

```
function prix(ht, tva):
{
    var ttc = ht* (1+tva/100);
    return ttc;
}
```

5 est l'argument ou paramètre effectif.

Taper le code de la fonction `hello_world` et de la fonction `double`. Appeler ces fonctions dans le *shell*.

Quelle différence noter dans la construction et l'appel de ces deux fonctions ?

5. Expliquer les résultats suivant:

```
>>> x = double(5)
>>> x
10
```

```
>>> x = hello_world()
>>> x
None
```

6. Reprendre le code de votre fonction `carre` et écrire une fonction `carre` prenant un paramètre `longueur` qui dessinera un carré dont la mesure du côté est `longueur` en pixels) dès qu'elle sera appelée.

7. Améliorer cette fonction pour que votre carré soit colorié avec une couleur passée en paramètre. Adapter votre fonction `tetro_L` en conséquence.

CC-BY-SA

Exemple :

```
>>> carre(100, "blue")
```

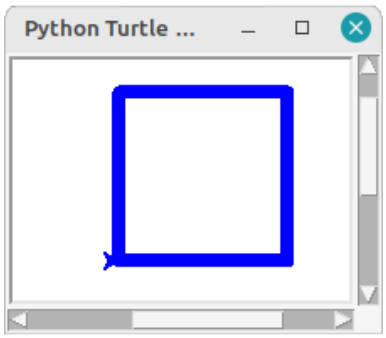


Figure 2: Utiliser plusieurs paramètres.

En bref Les **fonctions** sont des blocs d'instructions que l'ont a nommés: on peut les appeler dans le programme et ainsi **factoriser le code**. Ce bloc appelé **corps de la fonction** est délimité par différentes méthodes selon les langages: des accolades en C++ ou Javascript, l'indentation en Python.

Les fonctions peuvent avoir un ou plusieurs **paramètres** qui permettent de transmettre des valeurs à ce bloc : ces paramètres se comportent alors comme des variables connues seulement dans le corps de la fonction. De même, les variables déclarées dans le corps d'un fonction sont dites **locales** : elles ne sont connues que dans cette fonction. On dit que la **portée de la variable** est limitée à cette fonction : ces variables locales sont détruites dès la fin de l'exécution de la fonction. Le mot clé `return` permet de renvoyer une valeur : celle-ci pourra être utilisée ailleurs dans le programme, contrairement à une variable locale. On peut appeler une fonction pour l'exécuter ailleurs dans le programme, y compris dans une autre fonction, pourvu qu'elle ait été définie avant. Si des conditions portent sur les paramètres (préconditions) ou la valeur de retour (postconditions), elles doivent être documentées dans le corps de la fonction. On dit que l'on donne sa **spécification**.

Sa **signature** souvent donnée sur la première ligne est la donnée de son nom, du type de la valeur qui sera retournée et de ses paramètres.

Il est de plus nécessaire de tester le bon fonctionnement d'une fonction en prévoyant des **jeux de test**.

Syntaxe dans différents langages

En **C et C++**	En **Python**	En **Javascript**

#**include** <cmath>
float hypotenuse(**int** a, **int** b)
{
  return sqrt(a*a+b*b);
}
	from math **import** sqrt
	def hypotenuse(a, b):
	  return sqrt(a*a+b*b)
		function hypotenuse(a, b)
		{
		  return Math.sqrt(a*a+b*b);
		}

Notez l'import des modules0.5 de math en C et Python.

Exercice

1. Ecrivez une fonction `polygone` prenant `nb_cote`, `longueur` et `couleur` en paramètre et permettant de tracer un polygone à `nb_cote`, de longueur `longueur` et de couleur `couleur`.

2. Ecrivez une fonction `figure_carre` prenant un entier naturel `n` en paramètre et traçant `n` carrés en respectant la règle suivante :

- le premier carré mesure 5 pixels de côté,
- on décale de 5 degrés à chaque nouveau carré,
- pour chacun de ces carrés, la longueur du coté est augmentée de 5 pixels.

Note : Vous aurez besoin d'une variable qui joue le rôle d'**accumulateur** : la somme des longueurs des côtés y est stockée et s'incrémente à chaque tour de boucle.

0.3 Boucle non bornée

Lorsqu'on ne connait pas le nombre de tour, on peut répéter un bloc d'instructions tant qu'une *condition* est vérifiée: c'est la boucle **Tant que**

En **C** et **Javascript**	En **Python**

while (a⩽b)
{
  a = a+1;
}
	while a⩽b:
	  a = a+1

Une boucle *Tant que* est susceptible de ne pas s'arrêter : on dit qu'elle diverge. Sauf dans certain cas où on peut en avoir besoin, cela pose problème pusique la boucle se poursuit à l'infini.

Une technique pour prouver que cette boucle termine est celle du **variant de boucle.**

Variant de boucle Quantité entière, positive dont dépend l'arrêt de la boucle et qui décroit vers zéro. Dans l'exemple ci-dessus, la quantité $b - a$ est un variant de boucle, ce qui assure sa **terminaison**.

Exercices

1. Ecrire une fonction spirale qui trace une spirale carrée de longueur 500 pixels. La longueur de chaque nouveaux segments augmente de 10 pixels.
2. En utilisant la méthode `circle` du module `Turtle` , écrire une fonction `spirale_circ` traçant une spirale circulaire ayant pour longueur 1500 pixels, la longueur de chaque nouveau rayon augmente de 60 %. Utilisez la fonction `help` pour obtenir la documentation de la mléthode `circle`.

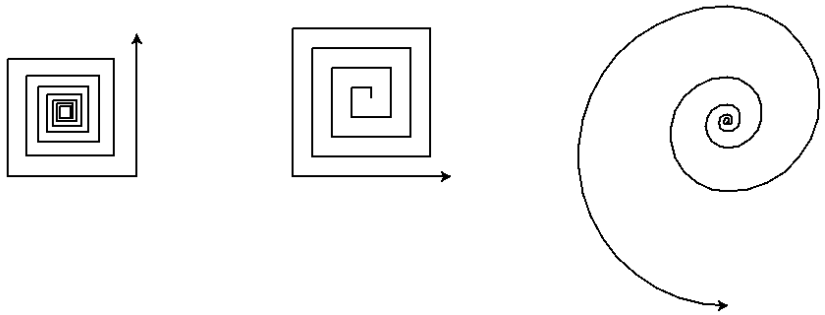


Figure 2: Exemples de réalisations

0.4 Structures conditionnelles

Pour tester une condition on utilise l'instruction `if`, éventuellement suivi d'un `else`:

En **C** et **Javascript**	En **Python**

if (a==b)
{
  a = b+1;
}
else
{
  a = a+1;
}
	if a == b :
	  a = b+1
	else:
	  a = a+1

On peut bien sûr avoir plus de deux cas à tester, voir la syntaxe dans le document de cours Inititation à Python 3:

1. Ecrire une fonction `maxi` qui prend en paramètre deux réels et renvoie la plus grande valeur entre a et b.
2. Ecrire une fonction `maxi3` qui prend en paramètre trois réels et renvoie la plus grande valeur entre a, b et c:
   (a) En utilisant la fonction `maxi`.
   (b) Sans utiliser la fonction `maxi`.

0.5 Compléments

Module Il existe plusieurs façon d'importer une bibliothèque de fonctions rangées dans un fichier ou module. Voir *Introduction à Python3*:
`https://maths-code.fr/cours/premiere-nsi-2/Cours-Python.pdf`.

Bonne pratiques Vous veillerez à respecter la PEP 8 : bonnes pratiques pour écrire un code lisible et homogène.

Exercices supplémentaires

1. Écrire une fonction `couleur_alea` qui renvoie unE couleur au format `(R,G,B)` aléatoire. Utilisez le module `random` pour obtenir un nombre de façon aléatoire.

   Exemple

   ```
   >>>couleur_alea()
   (18, 200, 144 )
   ```

2. Ecrire une fonction `frise` qui prendra en paramètre un entier `n` et tracera une frise avec n éléments. Construire une frise avec des carrés ou tout autre figure de couleurs différentes. Cette fonction utilisera les fonctions `polygone` et `couleur`. Vous pouvez écrire une fonction `avancer` prenant un paramètre `nb_pixel` qui permettra d'avancer de `nb_pixel` sans tracé.
3. Construire une face de dé affichant 4 points.

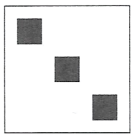
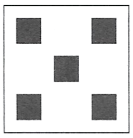
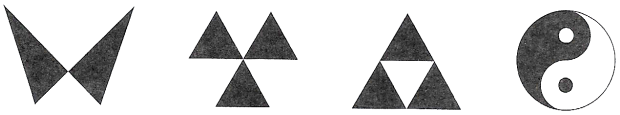



Figure 3: Exemples

4. Autres réalisations possibles.



0.6 Un mini-projet

Ecrire un programme Python réalisant une image. Vous veillerez à respecter la PEP 8 et à factoriser votre code en écrivant un maximum de fonctions. Vous trouverez des aides supplémentaires sur `https://nsi.xyz/art/`).

- Documentation `Turtle`.
- Exporter une image, pour pouvoir enregistrer l'image.

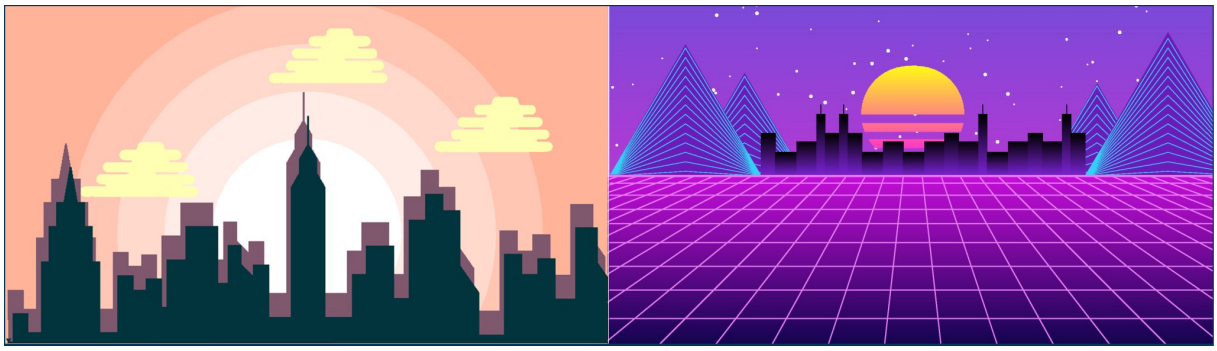


Figure 3: A vous de jouer !

0.7 Exercices

Exercice 1

1. Ecrivez une fonction `somme(n, m)` prenant deux nombres en paramètre et qui renvoie la somme des entiers compris entre n et m inclus.

   Exemple :

   ```
   >>> somme(1, 3)
   6
   >>> somme(5, 8)
   26
   ```

2. La population d'un village de 1000 habitants augmente de 8 % chaque année à partir de 2020, programmer une fonction `simulation` prenant un parametre `seuil` entier qui renvoie le nombre d'année nécessaire pour dépasser le seuil d'habitant passé en argument.

3. En réutilisant l'activité, on souhaite écrire une fonction `zigzag` qui prend en paramètre un entier n et trace une frise de n éléments.

   Cette frise s'orientera à gauche de son tracé lorsque le nombre d'éléments sera divisible par 5, et à droite sinon.

Exercice 2 Voici deux fonctions écrite en C++. Il n'est bien entendu aps questions ici d'apprendre ce langage mais de lire le code et reconnaître la syntaxe et les différentes structures.

1. A quoi sert le mot clé `void` ?
2. Que signifie le mot clé `int` ? A quoi sert-il ici ?
3. A quoi servent les accolades ?
4. Que font les `fonction1` et `fonction2` ?

```
void fonction1(int *tableau, int i, int j)
{
        int temp = tableau[i];
        tableau[i] = tableau[j];
        tableau[j] = temp;
}

void fonction2(int *tableau,int min_indice,int max_indice)
{
        int i=min_indice;
        for (int j=min_indice+1; j<=max_indice; j++)
        {
                if  (tableau[j] < tableau[i])
                    i = j;
        }
            return i;
}
```

0.8 Références

- *Numérique et sciences informatiques (Balabonski, Conchon, Filliâtre, Nguyen).*
- Le site `https://nsi.xyz/art/)` (Merci à Vincent Robert et ses élèves pour leur travail).