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 α° 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`.

Exemple: Python

```
for i in range (1, 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 3 fois 'Vive la NSI' ?
2. Reprenez vos réponses aux questions 1 et 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**(1,11):
	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 le monde !")` 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.

A faire:

1. Reprendre le code de votre carré et écrire une fonction `carre()` qui dessinera votre carré dès qu'elle sera appelée.
2. De même, écrivez une fonction `hexagone()` et `decagone()`.
3. Nous aimerions dessiner des polygone 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 en Python et 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(x)`. Appeler ces fonctions dans le *shell*.

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

4. Expliquer les résultats suivant:

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

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

5. Reprendre le code de votre carré et écrire une fonction `carre(longueur)` qui dessinera un carré de côté `longueur` dès qu'elle sera appelée.
6. Améliorer cette fonction pour que votre carré soit colorié avec une couleur passée en paramètre.

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. Le mot clé `return` permet de renvoyer une valeur: celle-ci pourra être utilisée ailleurs dans le programme, contrairement à une variable locale.

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.

Exercices

1. Ecrivez une fonction `polygone(nb_cote, couleur)` permettant de tracer un polygone à `nb_cote`, et de couleur `couleur`.
2. Ecrivez une fonction `somme(n, m)` qui renvoie la somme des entiers compris entre n et m inclus.

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**

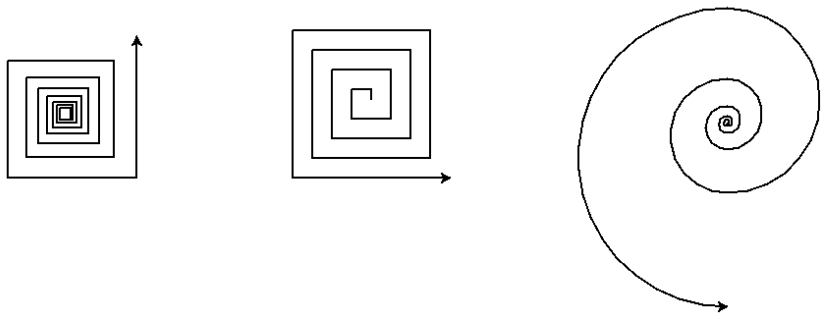
while (a⩽b)
{
  a = a+1;
}
	while a⩽b:
	  a = a+1

Une boucle *Tant que* est susceptible de ne pas s'arrêter, de diverger. 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. La population d'un village de 1000 habitants augmente de 8 % chaque année à partir de 2020, programmer une fonction `simulation(seuil:int)->int` qui retourne le nombre d'année nécessaire pour dépasser le seuil d'habitant passé en argument.
2. Tracer une spirale carrée de longueur 500 pixels. La longueur de chaque nouveaux segments augmente de 10 pixels.
3. En utilisant la méthode `circle()`, tracer une spirale ayant pour longueur 1500 pixels, la longueur de chaque nouveau rayon augmente de 10 % .



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 python:

`http://maths-code.fr/NSI/1ere/Cours-Python_2020.pdf`

1. Ecrire une fonction `maxi(a, b)` qui prend en paramètre deux réels et retourne la plus grande valeur entre a et b.
2. Ecrire une fonction `maxi3(a, b, c)` qui prend en paramètre trois réels et retourne la plus grande valeur entre a, b et c.
3. En réutilisant l'activité, écrire une fonction `zigzag(n)` qui prend en paramètre un entier n et retourne 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.

0.5 Compléments

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

1. Pour les plus rapides, écrire une fonction `couleur_alea()` qui renvoie un couleur au format `(R,G,B)` aléatoire.

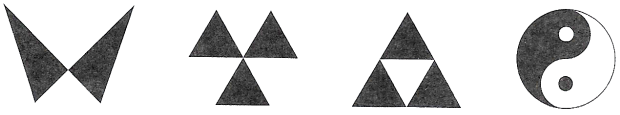
   Exemple

   ```
   >>>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()`.
3. Construire une face de dé affichant 4 points.

Exemples

4. Autres figures



0.6 Exercices

Voici deux fonctions écrite en C++.

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;
}
```

0.7 Références

- *Numérique et sciences informatiques (Balabonski, Conchon, Filliâtre, Nguyen).*