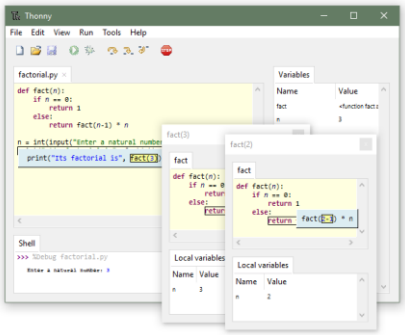
NSI	Langages et programmation Prise en main	TP – **Prise en main de l'EDI et premier programme**



Objectifs :

- ⇨ Savoir ce qu'est un EDI et savoir utiliser les fonctions de base de l'EDI Thonny
- ⇨ Découvrir les concepts de base des langages de programmation
- ⇨ Apprendre les éléments de syntaxe de base de python
- ⇨ Ecrire des programmes simples utilisant la bibliothèque `turtle`

I - L'EDI

1) Qu'est-ce qu'un EDI ?

Un EDI (**E**nvironnement de **D**éveloppement **I**ntégré) ou IDE en anglais est un logiciel qui réunit plusieurs outils nécessaires aux informaticiens pour créer des programmes.
Les fonctions de base d'un EDI sont donc :

- L'éditeur de texte qui permet de saisir le code source
- Le module d'exécution qui comprend le compilateur et/ou l'interpréteur et permet de tester le programme
- Le débogeur (debugger en anglais) qui permet l'exécution pas à pas du programme et aide à trouver les erreurs

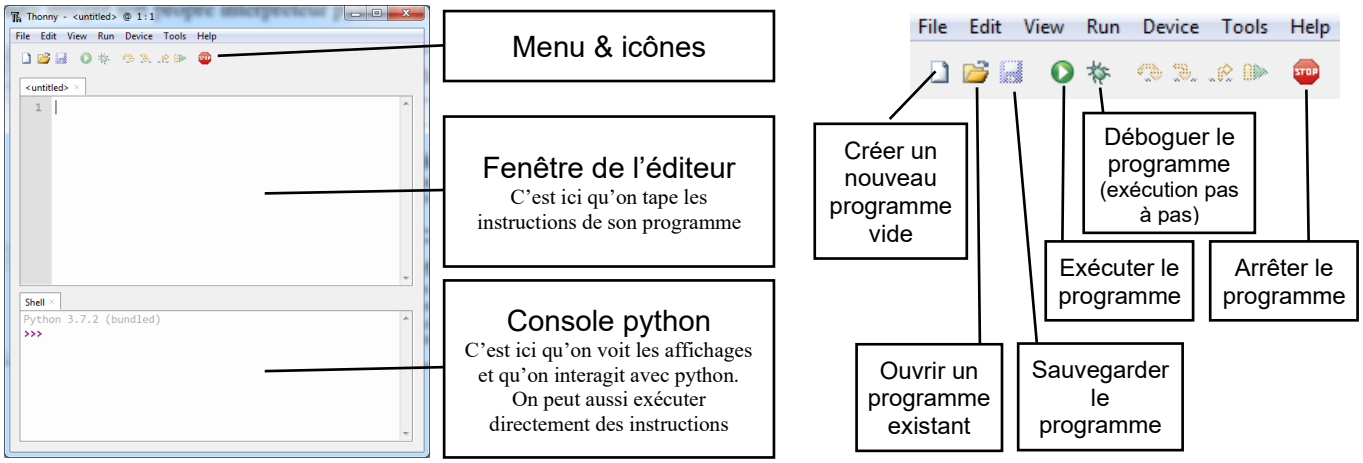
Certains EDI permettent d'aller plus loin en offrant des fonctions d'aide, la complétion automatique, un système de versionnage, de reformatage du code, de détection d'erreur, le support de plusieurs langages, …

2) Un EDI adapté aux débutants

Cette année, nous commencerons avec un EDI pour python qui est assez simple et pédagogique : Thonny
Il est téléchargeable sur le site officiel : https://thonny.org/ (il existe une version portable particulièrement utile si on veut pouvoir développer sur n'importe quel ordinateur).

Thonny intègre son propre interpréteur python. Il n'est donc pas nécessaire d'installer python séparément.
On utilisera la version 3.7 de python[1].

Aperçu de l'interface de Thonny :



[1] Ceci a son importance car les versions 2.x et 3.x de python sont incompatibles : un programme écrit pour python 2.1 peut ne pas s'exécuter correctement avec l'interpréteur python 3.5 par exemple.

Cette interface permet d'utiliser python de deux façons différentes :

- **En mode interactif**

On utilise alors la console python (partie du bas). Celle-ci affiche l'*invite de commande* (« `>>>` »). On peut alors taper une instruction et lorsqu'on appuie sur « Entrée », celle-ci est immédiatement exécutée et si elle produit un résultat, celui-ci est automatiquement affiché[2].
Ce mode est adapté pour faire des essais et comprendre le comportement du langage.

- **En mode script (ou « programme)**

On utilise cette fois la fenêtre de l'éditeur (en haut de l'interface) et on peut écrire une suite d'instructions qui seront enregistrées dans un fichier texte avec l'extension « `.py` ». Pour exécuter ces instructions, il faudra cliquer sur l'icône verte d'exécution, choisir l'option de menu « Run/Run current script » ou faire F5.
Ce mode est adapté pour écrire des programmes. C'est celui qu'on utilisera presque tout le temps.

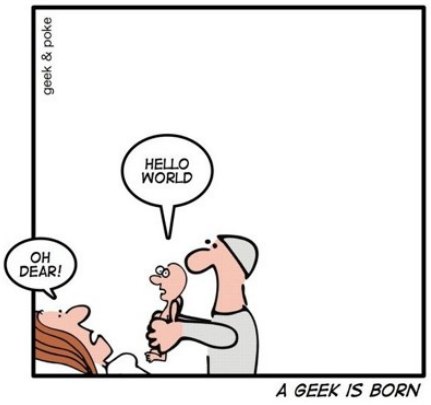
II – Premier programme : Hello world

1) Avant de commencer

Les programmes python doivent suivre quelques règles de présentation :

- Python est sensible à la casse. C'est-à-dire qu'il fait la différence majuscule/minuscule. Ainsi pour lui print, Print et PRINT sont trois expressions différentes (et seule la première correspond à une fonction de python).
- Une ligne de programme se termine au saut de ligne : chaque ligne de texte du fichier correspond à une ligne du programme[3].
- On peut rajouter des espaces dans le programme pour aérer le texte et le rendre plus lisible, **mais pas en début de ligne[4]**.

2) Bonjour le monde !



Il est de tradition chez les informaticiens de commencer l'apprentissage d'un nouveau langage par un programme qui ne fait qu'afficher une phrase à l'écran : c'est le programme « Hello world ! ».
L'instruction qui affiche du texte dans la console est dans presque tous les langages la commande `print`.
Comme toutes les fonctions, il faut lui fournir entre parenthèses les *arguments*, c'est-à-dire les données avec lesquelles elle doit travailler. Ici on fournit en argument le texte à afficher.
Là aussi c'est une quasi-constante dans tous les langages, les textes (ou *chaînes de caractères*) doivent être mis entre guillemets simples `'` (touche '4' du clavier) ou doubles américain `"` (touche '3' du clavier).

[2] Un tel fonctionnement est appelé « REPL » (**R**ead **E**valuate **P**rint **L**oop) il est commun à plusieurs interface de programmation comme l'invite de commande ou le terminal.
[3] Il est cependant possible de scinder une ligne de programme particulièrement longue en plusieurs ligne de texte en mettant le caractère '\' comme dernier caractère de la ligne à continuer.
[4] On verra plus tard que l'endroit où commence la ligne sert à python pour repérer les *blocs d'instructions*.

Q1 : Ecrire un programme « Hello_world.py » qui affiche « `Hello world !` », l'exécuter et vérifier qu'on obtient bien le résultat attendu.

III – Concepts de base

Maintenant que l'on sait écrire un programme et l'exécuter, on va s'en servir pour mettre en évidence les concepts de programmation.

Le principe de fonctionnement global de tous les langages est le même : à chaque ligne du programme la machine va chercher à ***évaluer*** l'expression qu'il rencontre. Pour ce faire il décompose la ligne en sous-expressions puis applique les opérateurs ou fonctions qu'il rencontre. Chaque fonction ou opérateur renvoi un résultat jusqu'à ce que l'évaluation globale de la ligne produise également un résultat (qui peut être `None`, c'est-à-dire rien d'utile en python).

Exemple :
Pour évaluer la ligne `2 * (3 - math.pow(5 + 2, 2))`
L'ordinateur décompose :

```
2 * (3 - math.pow(5 + 2, 2))
2 * (3 - math.pow(7, 2))
2 * (3 - math.pow(7, 2))
2 * (3 - 49.0)
2 * (3 - 49.0)
2 * (-46.0)
2 * (-46.0)
-92.0
```

La fonction `math.pow` correspond à la fonction puissance : `math.pow(a,b)` = a^b

Q2 :
1) Ecrire un programme « expression_Math.py » qui contienne la ligne vue en exemple et essayer de l'exécuter.

On obtient l'erreur « `NameError: name 'math' is not defined` ». Cela est dû au fait que l'expression utilise une fonction du module '`math`' et que l'on n'a pas indiqué à python qu'on souhaitait l'utiliser.

2) Rajouter l'instruction « `import math` » au tout début du programme (avant la ligne précédente) puis exécuter à nouveau le programme.
3) Cette fois le programme ne génère pas d'erreur, mais il ne produit aucun résultat visible. Pourquoi ? Modifier le programme pour qu'il affiche le résultat de l'opération.

On va demander à Thonny d'exécuter le programme pas à pas pour se rendre compte de la façon dont python exécute cette ligne. Pour ce faire, on doit cliquer sur l'icône de débogage :  (ou Ctrl-F5). Cliquer ensuite plusieurs fois sur l'icône  (*Step-into* – F7).

4) La décomposition faite par Thonny correspond-t-elle à celle vue plus haut ? Quelle est la valeur finale de l'évaluation de la ligne 2 ?

Une ligne de code est donc généralement constituée d'opérateurs, de fonctions et de données et elle est évaluée par l'interpréteur python qui la décompose en sous-expressions selon les règles mathématiques classiques (parenthèses et priorités des opérations).

1) Les opérateurs

Les opérateurs utilisables en python sont (liste volontairement non exhaustive que l'on sera amené à compléter plus tard) :

Les opérateurs de base :	Exemple
Opérations classiques : + ; – ; * (la multiplication est notée *) ; /	`2 * 3 + 5` vaut 11
Mise à la puissance : **	`4**3` vaut 4 à la puissance 3 donc 64
Division entière : //	`21 // 4` vaut 5
Reste de la division (utile pour les modulos ou pour la parité) : %	`21 % 4` vaut 1 (n'est pas un multiple de 4)

2) Les fonctions

On reconnait une fonction au fait qu'elle est constituée d'un **nom suivi d'une parenthèse ouvrante**. Suivent ensuite les **arguments**[5] de la fonction (les données à partir desquelles elle va travailler), séparés par des virgules. Puis la parenthèse fermante.
Lorsque python rencontre une fonction dans une expression, il va d'abord évaluer tous ses arguments, puis il va aller exécuter le bout de programme qui correspond à la fonction (et qui est stocké ailleurs en mémoire) en lui fournissant les arguments pour qu'il puisse remplir son rôle. La fonction renvoie une valeur[6] (appelée « *valeur de retour* ») qui peut être utilisée pour calculer une expression (comme la fonction `math.pow()` de l'exemple précédent).

Certaines fonctions ne sont décrites que dans des modules externes au programme que l'on appelle « bibliothèques » (« library » en anglais) ou modules. Si on souhaite les utiliser, il faut d'abord spécifier à python que l'on utilise la bibliothèque. Pour ce faire on doit importer la bibliothèque dans le programme python en utilisant l'instruction `import` suivie du nom de la bibliothèque (c'est ce que l'on a fait à la question 2 de l'exercice précédent). Pour utiliser les fonctions de la bibliothèque il faudra alors faire précéder le nom de la fonction du nom de la bibliothèque suivi d'un point. Exemple : `math.sin(0.7)` appelle la fonction `sin` du module `math`.

Python reconnait certaines fonctions « de base » pour lesquelles il n'est pas besoin d'appeler une bibliothèque. Ce sont les fonctions « built-in ». On peut les repérer au fait que Thonny les colore en violet dans le texte.
C'est le cas de la fonction `print()` vue précédemment.

Enfin il y a deux fonctions built-in particulières qui s'utilisent généralement en mode interactif (dans la console) et que nous allons présenter tout de suite :

- La fonction `dir()` permet de lister toutes les fonction du module donné en argument.
- La fonction `help()` qui affiche une aide (en anglais) sur la fonction donnée en argument.

```
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf',
'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'remainder',
'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
>>> help(math.pow)
Help on built-in function pow in module math:

pow(x, y, /)
    Return x**y (x to the power of y).
```

[5] Il se peut qu'il n'y ait aucun argument. Dans ce cas on doit quand même écrire les parenthèses pour indiquer qu'il s'agit d'une fonction. Exemple : `reset()`.
[6] Qui peut éventuellement être `None`, c'est-à-dire rien.

3) Les instructions du langage

Une ligne de code peut également contenir des instructions du langage. Ces instructions ne nécessitent généralement pas de parenthèses comme les fonctions et peuvent agir sur la façon dont se déroule le programme. Python les repère facilement car toutes ces instructions sont des **mots-clés** et il est interdit de les utiliser pour autre chose (comme un nom de fonction par exemple). Thonny les colore en violet et les met en gras pour les mettre en évidence.
Nous ne les verrons pas tous et surtout nous les découvrirons progressivement, mais il est utile d'en connaître la liste pour ne pas les utiliser pour autre chose dans nos programmes.

Liste des mots-clés de python :

and	**assert**	**break**	**class**	**continue**	**def**	**del**	**elif**	**else**
except	**exec**	**finally**	**for**	**from**	**global**	**if**	**import**	**in**
is	**lambda**	**not**	**or**	**pass**	**print**	**raise**	**return**	**try**
while	**yield**	**True**	**False**	**None**				

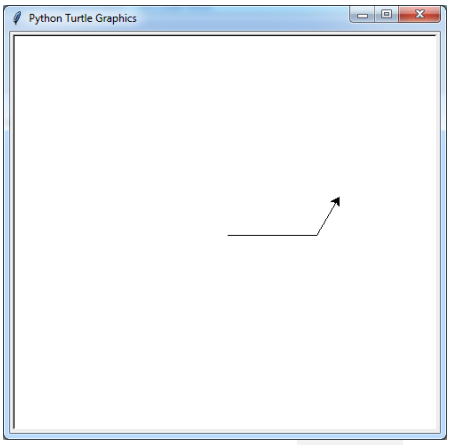
Pour l'instant nous avons juste utilisé l'instruction `import` qui permet de déclarer l'utilisation d'une bibliothèque de fonctions.

IV – Deuxième programme

Pour ce deuxième programme, nous allons utiliser les fonctions d'une bibliothèque permettant de réaliser des tracés facilement : la bibliothèque `turtle`.

Le principe de ce module est assez simple : on dispose d'un stylo (appelé « tortue » et représenté à l'écran par une petite flèche) que l'on va déplacer dans une fenêtre et qui va laisser derrière lui une trace exactement comme un stylo sur une feuille. On pourra ainsi tracer toutes sortes de formes.

Pour une utilisation plus facile de ce module, Thonny propose une option pour garder la fenêtre de turtle toujours au même endroit et à l'avant plan ce qui facilite la mise au point du programme car on peut voir le résultat du programme tout en le modifiant. Pour activer cette option, il faut la cocher dans le menu « Run/Dock user windows ».



Fenêtre de tracé de `turtle`

La commande `dir(turtle)` permet d'afficher toutes les fonctions du module, mais nous allons pour l'instant présenter seulement quelques fonctions utiles pour nous :

`reset()`	Création et initialisation de la fenêtre de tracé
`forward(x)`	Fait avancer la tortue de x pas (pixels) en avant
`backward(x)`	Fait reculer la tortue de x pas (pixels)
`goto(x, y)`	Déplace la tortue en ligne droite de là où elle se trouve vers le point de coordonnée (x, y).
`left(x)`	Fait tourner la tortue sur place vers la gauche d'un angle de x degré
`right(x)`	Fait tourner la tortue sur place vers la droite d'un angle de x degré
`setheading(x)`	Fixe l'orientation de la tortue à l'angle x (l'origine des angles est l'horizontale droite)
`up()`	Lève le stylo (permet de se déplacer sans tracer)
`down()`	Pose le stylo (on trace en se déplaçant)
`color("couleur")`	Change la couleur du stylo en la mettant à *couleur* où *couleur* est un mot anglais désignant la couleur comme red, yellow, blue, grey, black, green, … (voir la page https://ecsdtech.com/8-pages/121-python-turtle-colors pour une liste des couleurs)
`hideturtle()`	Cache la tortue (flèche). Tout ce qu'elle a tracé reste visible
`showturtle()`	Montre la tortue. Permet de visualiser où elle se trouve et son orientation
`done()`	Attend la fermeture de la fenêtre de `turtle`. Utilisé en général en fin de programme.

1) Première version

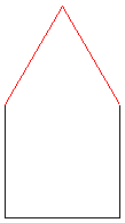
Q3 :
1) Quelle doit être la première ligne du programme pour pouvoir utiliser les fonctions du module turtle par la suite ?
2) Ecrire un programme qui trace un carré noir (couleur par défaut) de 100 pixels de côté. Enregistrer le programme sous le nom « tortue_v1.py ». Exécuter le programme et vérifier qu'on obtient bien le résultat attendu.

2) Deuxième version

Vu le nombre important d'appels aux fonctions de `turtle`, on va maintenant utiliser une astuce qui va nous permettre de simplifier l'écriture. Au lieu d'utiliser l'instruction `import turtle`, on va utiliser une de ses variantes :

`from turtle import *` : permet de faire référence à toutes les fonctions du module turtle directement sans avoir à préciser le module. Ainsi on pourra écrire `forward(100)` plutôt que `turtle.forward(100)`. Nous verrons dans le cours qu'il faut cependant se méfier de cette façon de faire.

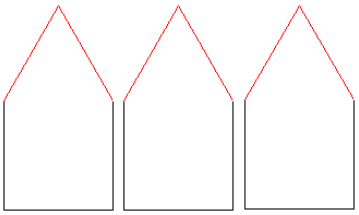
Q4 : Modifier le programme précédent pour qu'il dessine une maison constituée d'un carré noir de 100 pixels de côté et d'un « toit » sous la forme d'un triangle rouge posé sur le carré précédent. Enregistrer le programme sous le nom « tortue_v2.py ».



3) Troisième version

On veut maintenant représenter un village.

Q5 : Modifier le programme précédent pour qu'il dessine 3 maisons côte à côte espacées de 10 pixels. Enregistrer le programme sous le nom « tortue_v3.py ».



On s'aperçoit d'une part qu'il a fallu rajouter des instructions pour replacer la tortue correctement entre chaque dessin de maison et d'autre part qu'on a écrit plusieurs fois le même code ce qui est fastidieux et peut être source d'erreur. Si on devait représenter 10 ou 100 maisons, on voit que la méthode de recopie du code n'est pas gérable… Nous verrons lors du prochain cours comment remédier à cela.

TRAVAIL A FAIRE POUR LA PROCHAINE SEANCE :

Reprendre le travail de la question 4 mais en améliorant le tracé de la maison : faire des formes remplies (voir les fonctions `begin_fill()` et `end_fill()` de turtle), rajouter des éléments (porte, fenêtres, …).