

R1.01 INITIATION AU DÉVELOPPEMENT

FEUILLE DE TP N°2

Prise en main de VSCode – Variables/environnement/fonctions

Qu'est-ce qu'on fait aujourd'hui ?

Cette feuille a pour objectifs principaux :

- découvrir l'EDI (Environnement de Développement Intégré) Visual Studio Code
- utiliser cet environnement pour manipuler la notion d'environnement d'un programme
- écrire les premiers programmes

Exercice 1 *Prise en main de VSCode*



EDI ? A quoi ça sert ?

L'utilisation de l'interprète Python en mode interactif n'est pas standard pour développer en Python. Un programmeur Python a besoin d'un éditeur de texte pour écrire ses scripts (programmes) et d'un interprète Python pour les exécuter. De manière générale un programmeur va utiliser un Environnement de Développement Intégré qui va lui fournir l'éditeur, l'interprète et plein d'autres choses dans un seul environnement.

L'EDI choisi à l'IUT'O est Visual Studio Code.

1.1 Lancement

Dans un terminal tapez les commandes suivantes permettant de

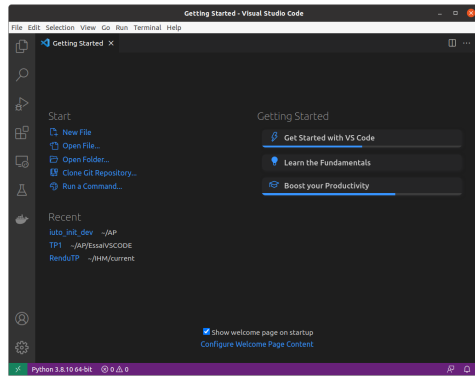
1. créer les répertoires de travail pour ce module,
2. d'installer les extensions utiles à la programmation Python
3. lancer VSCode

```
mkdir INIT_PROG
mkdir INIT_PROG/TP
code --install-extension ms-python.python
code --install-extension njpwerner.autodocstring
code
```

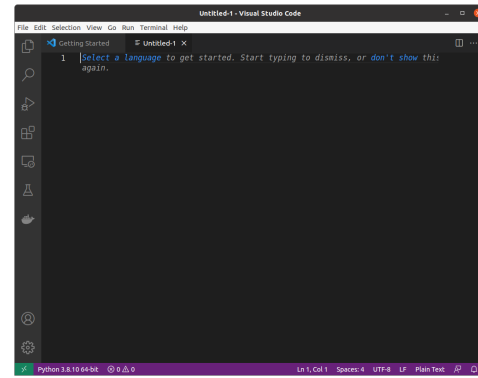
L'écran d'accueil de VSCode est donné figure 1a. Cliquez sur *Open Folder...* et choisissez le répertoire `~/INIT_PROG/TP`. Cliquez sur *Yes, I trust the authors*, puis sur *New file*. Le langage de votre fichier vous sera demandé (figure 1b) : sélectionnez *Python*.

Ecrivez le programme

```
1 print("Bienvenue à l'IUT'O")
```



(a) Ecran d'accueil VSCode



(b) Choix du langage

FIGURE 1

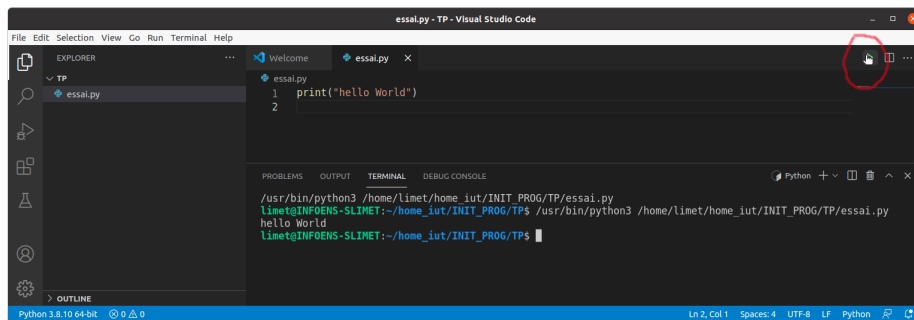


FIGURE 2 – Exécution d'un script Python

Cliquez sur *File -> Save* et choisissez `essai.py` comme nom de fichier. Ce fichier doit apparaître dans l'explorateur sur la gauche de l'écran.

Pour exécuter ce superbe programme il suffit de cliquer sur la flèche verte en haut à droite de la fenêtre VSCode.

Un nouveau panneau apparaît sous l'éditeur de texte (figure 2) qui affiche la commande lancée par VSCode pour exécuter votre programme et affiche le message **Bienvenue à l'IUT'0**. Dans l'onglet *Terminal*, vous pouvez taper des commandes **bash** comme par exemple `python3 essai.py` pour lancer votre programme.

Exercice 2 Utilisation du débogueur

Qu'est qu'un débogueur

La plupart du temps, les programmeurs n'arrivent pas à programmer correctement dès le premier jet. Afin de les aider à retrouver leurs erreurs, les EDI fournissent des débogueurs qui permettent d'exécuter le programme pas à pas et d'observer l'évolution de l'environnement.

Créez le fichier `exo_debug.py` et implémentez le programme suivant :

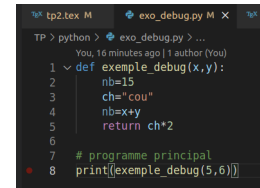
```
def exemple_debug(x, y):
    nb = 15
    ch = "cou"
    nb = x+y
    return ch*2

# programme principal
print(exemple_debug(5, 6))
```

2.1 Exécutez le script. Qu'affiche-t-il ?


2.2 Poser un point d'arrêt (breakpoint)

Afin de déboguer notre code, il faut dans un premier temps dire à partir de quel endroit on souhaite exécuter le programme instruction par instruction (pas à pas). C'est la notion de *point d'arrêt* ou *breakpoint*. Pour poser un point d'arrêt sur une ligne du programme, il suffit de cliquer sur la 1ère colonne de cette ligne (voir figure). Un point rouge apparaît.



Pour enlever un point d'arrêt, il suffit de cliquer sur le point rouge et celui-ci disparaît.

2.3 Exécuter en mode debug

Pour exécuter un programme en mode débogage il faut cliquer sur l'icône , puis sur *Run and Debug* et choisir l'option *Python File*. Vous devez obtenir l'écran de la figure 3.

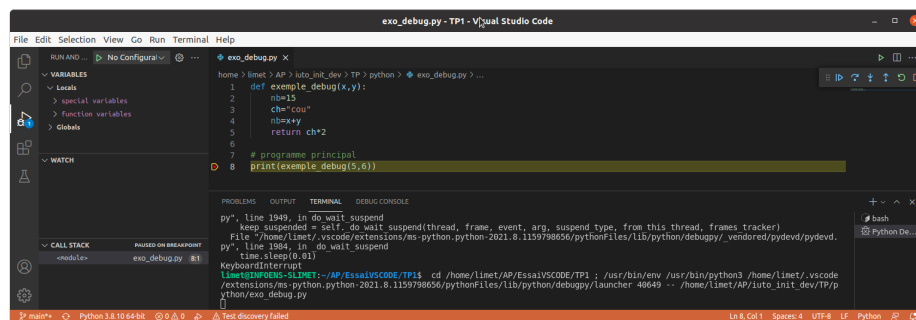


FIGURE 3 – Débogage d'un script Python

Une fois le débogger en marche, plusieurs possibilités de poursuivre l'exécution du programme sont disponibles via la boîte à outils . Dans l'ordre de gauche à droite :

- **Continue** : reprendre l'exécution jusqu'au prochain point d'arrêt ou jusqu'à la fin du programme.
- **Step over** : exécuter la prochaine ligne de code, sans entrer dans les appels de fonction.
- **Step into** : exécuter la prochaine ligne de code en entrant à l'intérieur des fonctions.
- **Step out** : finir l'exécution de la fonction en cours et retourner à la ligne d'appel de la fonction.
- **Restart** : interrompre l'exécution en cours et réinitialiser le processus de débogage.
- **Stop** : interrompre le processus de débogage en cours.

2.4 Environnement : contenu des variables

Lorsque le processus de débogage est lancé, apparaît sur la gauche de l'écran un panneau d'observation de l'environnement du programme (figure 2). La section *VARIABLES* permet de voir les variables définies dans l'environnement. Cette section contient deux sous sections *Locals* qui est l'environnement de la fonction en cours d'exécution et *Globals* qui est l'environnement global du programme. Ces deux environnements sont identiques lorsqu'on est dans le programme principal.

Avec le débogger, entrez à l'intérieur de la fonction `exemple_debug`. On voit apparaître la création de l'environnement de la fonction avec les deux paramètres `x` et `y` initialisés avec la valeur des paramètres réels au moment de l'appel (figure 4).

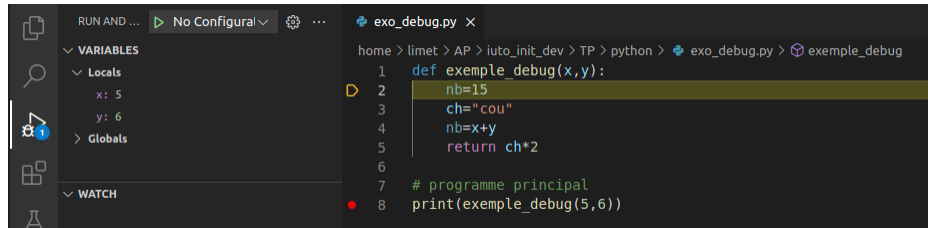


FIGURE 4 – Environnement local d’une fonction

Si vous exécutez le code de la fonction pas à pas avec **Step into** vous voyez l’environnement de la fonction évoluer. Remarquez que l’on peut voir le type des variables en passant la souris sur son nom dans *Locals*. Par ailleurs, lorsque la fonction se termine, on voit la valeur de retour.

Exercice 3 *Environnement, variables etc...*

3.1 En utilisant le débogueur, décrivez l’environnement de la fonction en cours d’exécution (en donnant explicitement le type des variables).

| Programme | Environnement | Votre commentaire |
|---|----------------------------|--------------------|
| <pre>def fonction1(x): a = 12 a = 3+y return a print(fonction1(23))</pre> | <pre>a = int x = int</pre> | y n'est pas défini |

3.2 En utilisant le débogueur, décrivez l’environnement de la fonction avec l’appel `fonction2(5)` en cours d’exécution (en donnant explicitement le type des variables).

| Programme | Environnement | Votre commentaire |
|--|--|-------------------|
| <pre>def fonction2(x): a = 12 a = 3+x A = a/2 b = a a = 5 b = b+1 return a print(fonction2(5))</pre> | <pre>a = int A = float b = int x = int</pre> | |

Exercice 4 *Documentation de code Traduction algo → Python*

4.1 Documentation docstring

Voici un algorithme et sa traduction en Python

```
algo: sante
parametres:
    taille un nombre indiquant la taille de la personne en mètres
    poids un nombre indiquant le poids de la personne en kg
résultat: un chaine indiquant le problème éventuel de poids
debut:
    mettre poids/(taille*taille) dans imc
```

```

    si imc inférieur à 16.5
        mettre "famine" dans res
    sinon si imc inférieur à 18.5
        mettre "maigreur" dans res
    sinon si imc inférieur à 25
        mettre "normal" dans res
    sinon si imc inférieur à 30
        mettre "surpoids" dans res
    sinon
        mettre "obésité" dans res
    retourner res

```

```

def sante(taille, poids):
    """Permet de détecter un pb de santé en fonction de l'imc

    Args:
        taille (float): la taille de la personne en mètres
        poids (int): le poids de la personne en kg

    Returns:
        str: le problème éventuellement détecté
    """
    imc = poids/(taille*taille)
    if imc < 16.5:
        res = "famine"
    elif imc < 18.5:
        res = "maigreur"
    elif imc < 25:
        res = "normal"
    elif imc < 30:
        res = "surpoids"
    else:
        res = "obésité"
    return res

```

Cette fonction indique les problèmes de santé potentiel d'une personne en fonction de son IMC (Indice de Masse Corporelle) qui est calculé en divisant la taille de la personne par son poids au carré. Par exemple une personne mesurant 1.80m et pesant 80kg aura un IMC de 24.6914 qui est normal, alors qu'une personne ayant un poids de 67kg et une taille de 1.50m aura un IMC de 29.777 qui indique un surpoids.

4.2 Génération docstring

Créez un fichier `sante.py` et écrivez uniquement les lignes

```

def sante(taille, poids):

    return res

```

Puis placez le curseur sur la ligne vide, cliquez sur le bouton de droite de la souris et choisissez `Generate Docstring`. Vous devez obtenir

```
def sante(taille, poids):
    """_summary_
    Args:
        taille (_type_): _description_
        poids (_type_): _description_

    Returns:
        _type_: _description_
    """
    return res
```

Vous pouvez à présent compléter la documentation de la fonction

```
def sante(taille,poids):
    """Permet de détecter un pb de santé en fonction de l'imc

    Args:
        taille (float): la taille de la personne en mètre
        poids (int): le poids de la personne en kg

    Returns:
        str: le problème éventuellement détecté
    """
    return res
```

A partir de maintenant, lorsque vous tapez **sante** dans votre fichier ou que vous passez la souris sur ce mot, la documentation de la fonction s'affiche.

4.3 Tests de la fonction

Avant d'implémenter la fonction vous pouvez implémenter les exemples donnés plus haut sous forme de tests en ajoutant une fonction **test_sante** :

```
def test_sante():
    assert sante(1.8, 80) == "normal" #indique que sante(1.8, 80) doit retourner "normal"
    assert sante(1.6, 67) == "surpoids" #indique que sante(1.6, 67) doit retourner "surpoids"
```

4.4 Traduire le corps de la fonction.

4.5 Faire passer les tests

Si vous tapez dans l'onglet terminal de VSCode la commande :

```
1  pytest-3 -v sante.py
```

Vous obtenez le résultat des tests présents dans le fichier **sante.py**. Pour voir ce qu'il se passe en cas d'échec des tests vous pouvez ajouter un test erroné du type

```
assert sante(2.2, 40) == "normal"
```

IMPORTANT !

A partir de maintenant toutes vos fonctions devront être documentées par un docstring et avoir une fonction de test contenant au moins 4 **assert** !

Exercice 5 Traduire d'algo vers Python

Traduisez en Python les algorithmes suivants en documentant votre code. Testez vos fonctions avec différents paramètres. Exécutez ces fonctions avec le débogueur pour comprendre leur fonctionnement. Ecrivez une fonction de test permettant de vérifier vos hypothèses sur la fonction.

```
algo: algo1
    parametres: a,b,c,d des nombres
    résultat: (à vous de le définir)
    debut:
        si a est plus petit que b alors
            res=a
        sinon
            res=b
        si c est plus petit que res alors
            res=c
        si d est plus petit que res alors
            res=d
        retourner res

algo: algo2
    parametres: m un mot en minuscule
    résultat: (à vous de le définir)
    debut:
        mettre 0 dans res
        pour chaque lettre du mot
            si la lettre est dans 'aeiouy' alors
                ajouter 1 à res
            sinon
                retirer 1 à res
        retourner res>0
```

Exercice 6 Qualification aux jeux olympiques

La fédération d'athlétisme vient de publier les critères de qualification à l'épreuve du 100m des jeux olympiques. Ces critères sont les suivants :

- Pour les hommes
 - soit il faut avoir un record personnel au 100m inférieur à 12 secondes et avoir gagné au moins 3 courses dans l'année
 - soit être champion du monde de la discipline
- Pour les femmes
 - soit il faut avoir un record personnel au 100m inférieur à 15 secondes et avoir gagné au moins 3 courses dans l'année
 - soit être championne du monde de la discipline

6.1 Trouver quelles sont les données d'entrée qui permettent de prendre la décision.

h ou f, if h = pr100m < 12s and 3raceswin< in 1year or worldchampion. if f = pr100m < 15s and 3raceswin< or wc

6.2 Donner des exemples de personnes qualifiées, et d'autres non qualifiées.

6.3 Écrire l'algorithme qui permet de prendre la décision et implémentez-le en Python

Exercice 7 *Sécurité routière*

Consultez le site <https://www.controleradar.org/contraventions.html> afin de connaître la réglementation en terme de contravention en cas d'excès de vitesse.

- 7.1** Quels sont les paramètres qui permettent de déterminer les sanctions encourues ?
vitesse/ cmb de km/h supérieur a la limite et limite de vitesse

Le résultat de la fonction se compose de 3 entiers correspondant au montant de l'amende, au nombre de points perdus et au nombre d'années de suspension de permis.

- 7.2** Donner trois ou quatre exemples de sanctions en fonction de la valeur des paramètres d'entrée

- 7.3** Ecrire une fonction qui donne les sanctions encourues par un automobiliste en fonction des paramètre d'entrée. Le résultat sera un triplet d'entier



N'oubliez pas !

Il faut documenter la fonction avec un docstring et fournir une fonction de test avec les exemples que vous avez trouvés.