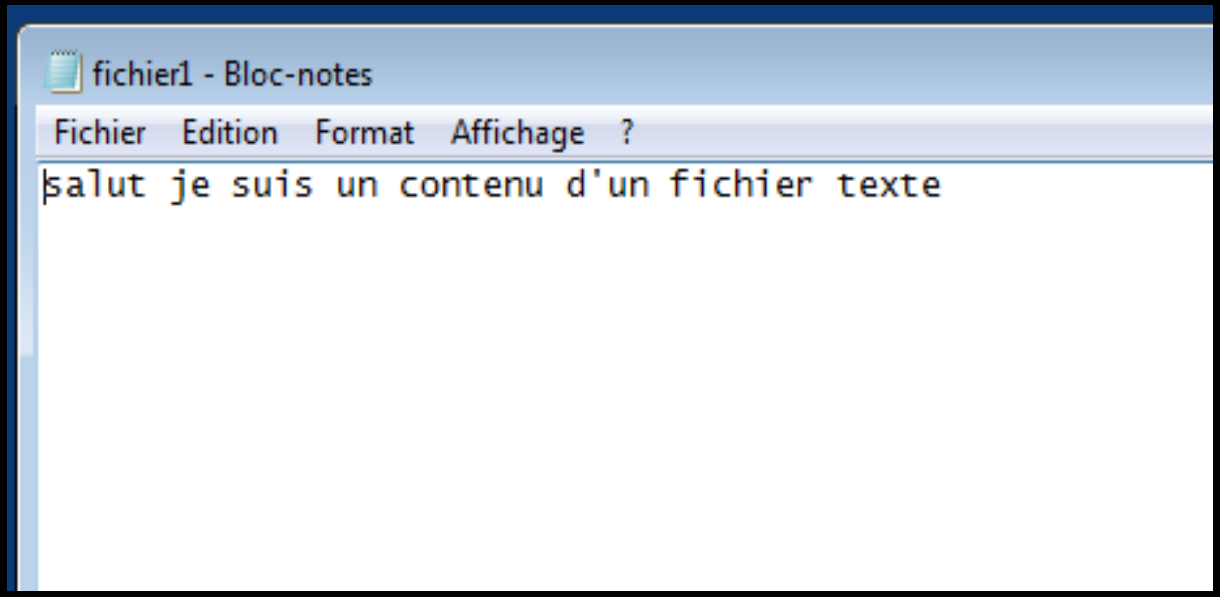
Les fichiers

Créer des petits morceaux de code et sans grand intérêt, le problème de l'interpréteur, c'est qu'une fois celui-ci fermé, votre travail est perdu.
L'idée d'un programme et fichier , c'est d'enregistrer votre travail dans un fichier et ensuite de l'exécuter.
Lorsque du code est enregistré dans un fichier exécutable on parle de **script** .

Ouvrir et fermer un fichier

Créant un fichier texte nommé "fichier1" sur le bureau

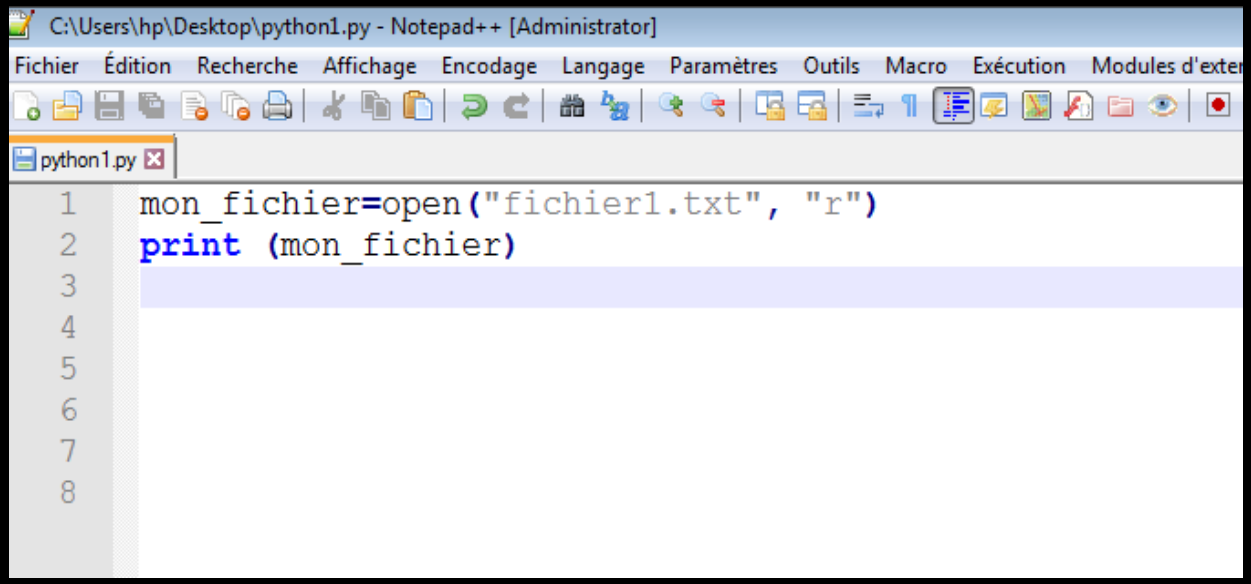


important

Pour créer un fichier vous utiliserez un nouveau fichier texte

Ouvrir et fermer un fichier

Créant un fichier python nommé
"python1" sur le bureau à l'aide de Notepad++ (ou autre éditeur)



La commande "open" permet d'ouvrir le fichier
L'argument "r" signifie en lecture seule
Le contenu est mis dans la variable "mon_fichier"

Cours réalisés par le professeur Khalid +212 622176552


```
C:\Users\hp>cd desktop

C:\Users\hp\Desktop>py python1.py
<_io.TextIOWrapper name='fichier1.txt' mode='r' encoding='cp1252'>

C:\Users\hp\Desktop>
```

L'exécution du fichier python1 avec affichage du contenu de la variable mon_fichier

Pour fermer le fichier on utilise la méthode "close()"

mon_fichier.close()

3 modes d'ouverture de fichiers

- `'r'` : ouverture en lecture (Read).
- `'w'` : ouverture en écriture (Write). Le contenu du fichier est écrasé. Si le fichier n'existe pas, il est créé.
- `'a'` : ouverture en écriture en mode ajout (Append). On écrit à la fin du fichier sans écraser l'ancien contenu du fichier. Si le fichier n'existe pas, il est créé.

Adressage relatif & absolu

Quand on décrit la position d'un fichier grâce à un chemin relatif, on tient compte du dossier dans lequel on se trouve actuellement.
Ainsi, si on se trouve dans le dossier C:\test et que l'on souhaite accéder au fichier fic.txt contenu dans ce même dossier, le chemin relatif menant à ce fichier sera tout simplement fic.txt.

Maintenant, si on se trouve dans C:, notre chemin relatif sera test\fic.txt.
Quand on décrit un chemin relatif, on utilise parfois le symbole .. qui désigne le répertoire parent.


```
mon_fichier=open("fichier1.txt", "r")
```

Remarquez l'utilisation de l'adressage relatif lors de la méthode open car le fichier d'exécution est dans le même répertoire que celui du fichier à ouvrir

Lire l'intégralité d'un fichier

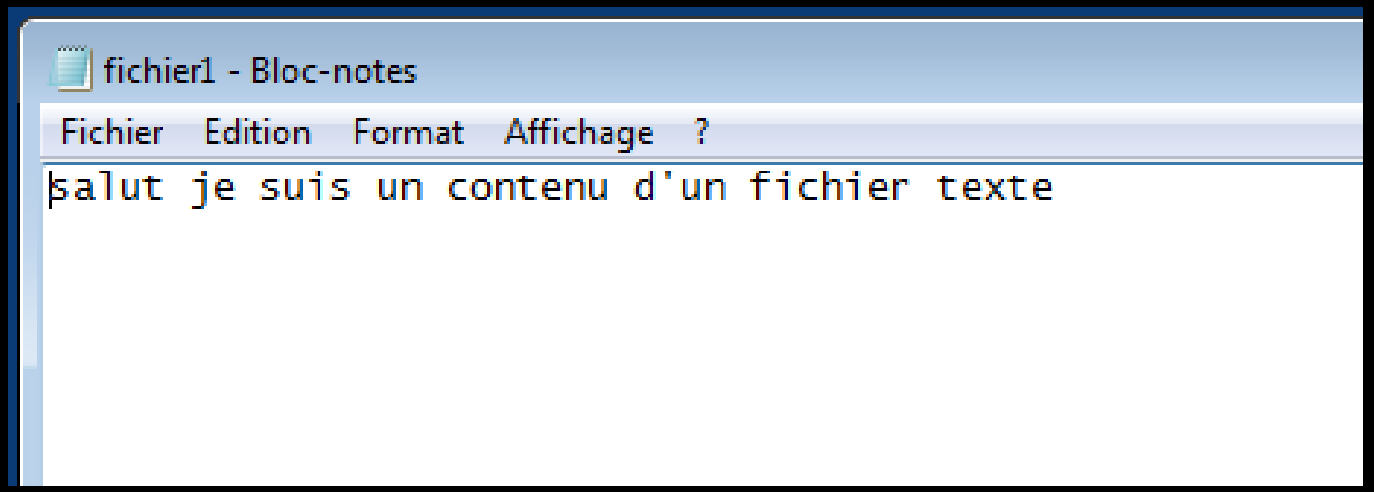
Méthode "read()"

Fichier python1

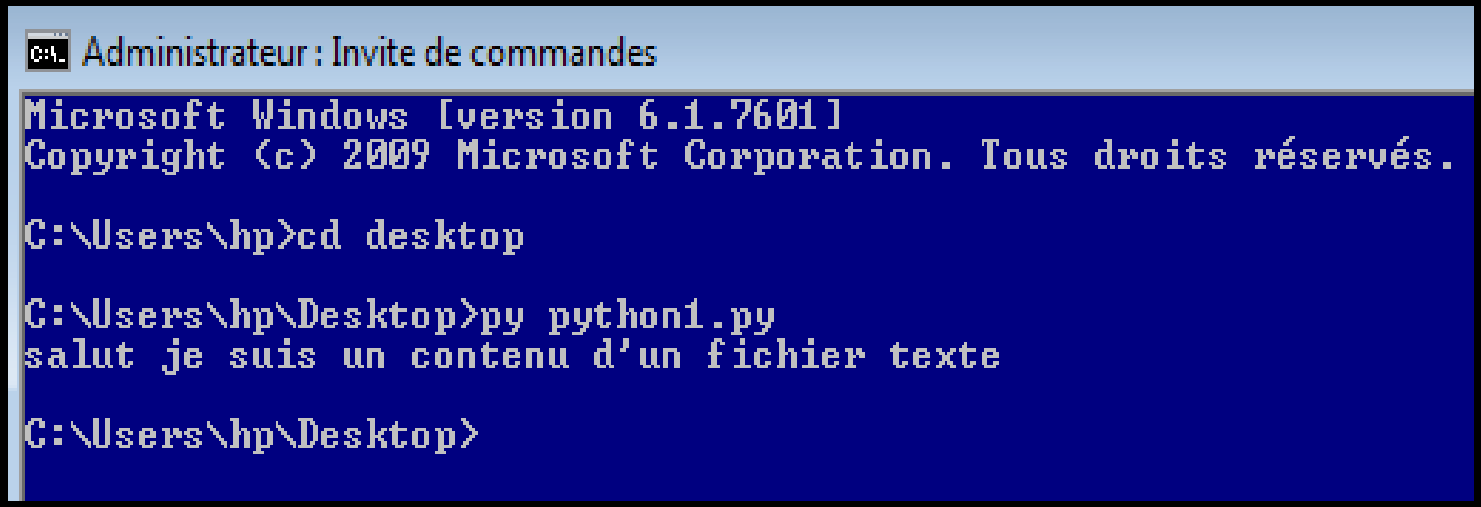
1.py

```
mon_fichier=open("fichier1.txt", "r")
contenu = mon_fichier.read()
print (contenu)
mon_fichier.close()
```

Fichier fichier1



Console



\n désigne un saut de ligne

Écriture dans un fichier

Méthode " write"

Bien entendu, il nous faut ouvrir le fichier avant tout.
Vous pouvez utiliser le mode w ou le mode a. Le premier écrase le contenu éventuel du fichier, alors que le second ajoute ce que l'on écrit à la fin du fichier.
Dans tous les cas, ces deux modes créent le fichier s'il n'existe pas.

Fichier python1

```
mon_fichier=open("fichier1.txt", "w")
mon_fichier.write("je suis un texte qui va écraser le contenu du fichier1")
print(mon_fichier)
mon_fichier.close()
```

Console

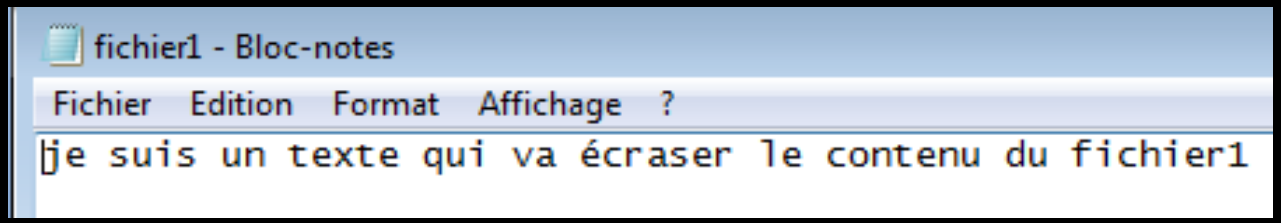
```
C:\Users\hp\Desktop>py python1.py
<_io.TextIOWrapper name='fichier1.txt' mode='w' encoding='cp1252'>

C:\Users\hp\Desktop>
```

important

La méthode write n'accepte en paramètre que des chaînes de caractères.

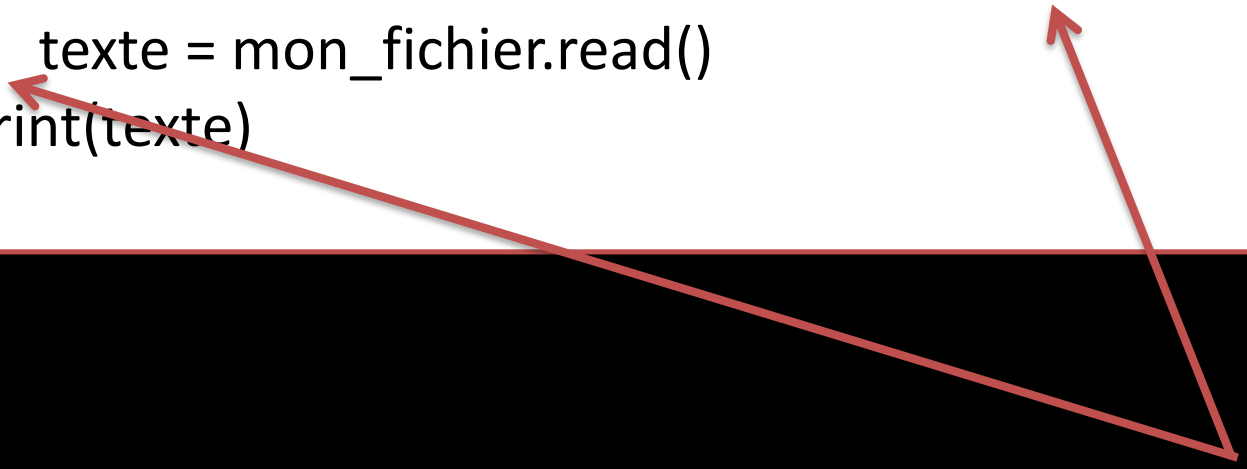
Fichier fichier1



Utilisation du mot clé "with"

```
with open(mon_fichier, mode_ouverture) as variable:
    # Opérations sur le fichier
```

```
>>> with open('fichier.txt', 'r') as mon_fichier:
        texte = mon_fichier.read()
>>> Print(texte)
```



Vous pouvez appeler `mon_fichier.closed` pour vérifier que le fichier est refermé. Si le fichier est fermé, `mon_fichier.closed` vaudra `True` .

Il est inutile, par conséquent, de fermer le fichier à la fin du bloc `with` . Python va le faire tout seul, qu'une exception soit levée ou non. Je vous encourage à utiliser cette

Les modules

Les modules sont des programmes Python qui contiennent des fonctions que l'on est amené à réutiliser souvent (on les appelle aussi bibliothèques ou librairies).

Les développeurs de Python ont mis au point de nombreux modules qui effectuent une quantité phénoménale de tâches qu'il suffit d'appeler par code pour les utiliser

Importer un module

Dans cet exemple nous appellerons random qui est un module Python regroupant plusieurs fonctions permettant de travailler avec des valeurs aléatoires.

La distribution des nombres aléatoires est réalisée par le générateur de nombres pseudo-aléatoires Mersenne Twister, l'un des générateurs les plus testés et utilisés dans le monde informatique.


```
>>> import random              # importation du module
>>> random.randint(0,10)      # appel de la fonction "randint"
4
```

cette fonction randint() renvoie un nombre entier aléatoirement tiré entre a inclus et b inclus (pas comme range(a,b)

Autre moyen d'importation d'une fonction

```
>>> from random import randint
>>> randint(0,10)
7
```

Importer tout d'un module

```
>>> from random import *
>>> x = [1, 2, 3, 4]                # définition d'une liste x
>>> shuffle(x)                      # mélange aléatoire des éléments de x
>>> x
[2, 3, 1, 4]
>>> shuffle(x)
>>> x
[4, 2, 1, 3]
>>> randint(0,50)
46
```

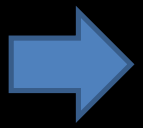
Privilégiez la première méthode

Obtenir de l'aide sur un module importé

```
>>> import random
>>> help(random)
```

On peut se déplacer dans l'aide avec les flèches ou les touches page-up et page-down .
Il est aussi possible d'invoquer de l'aide sur une fonction particulière d'un module de la manière suivante
help(module.fonction)

La commande help() est en fait une commande plus générale permettant d'avoir de l'aide sur n'importe quel objet chargé en mémoire.



```
>>> t = [1, 2, 3]
>>> help(t)
Help on list object:
class list(object)
 |  list() -> new list
 |  list(sequence) -> new list initialized from sequence's
items
 |
 |  Methods defined here:
 |
 |  __add__(...)
 |      x.__add__(y) <==> x+y
 |
...
```

Si on veut connaître d'un seul coup d'oeil toutes les méthodes ou variables associées à un objet, on peut utiliser la fonction dir()

```
>>> import random
>>> dir(random)
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST',
'SystemRandom',
'TWOPI', 'WichmannHill', '_BuiltinMethodType', '_MethodType', '__all__',
'__builtins__',
'__doc__', '__file__', '__name__', '_acos', '_ceil', '_cos', '_e', '_exp', '_hexlify',
'_inst', '_log', '_pi', '_random', '_sin', '_sqrt', '_test', '_test_generator',
'_urandom', '_warn', 'betavariate', 'choice', 'expovariate', 'gammavariate', 'gauss',
'getrandbits', 'getstate', 'jumpahead', 'lognormvariate', 'normalvariate',
'paretovariate', 'randint', 'random', 'randrange', 'sample', 'seed', 'setstate',
'shuffle', 'uniform', 'vonmisesvariate', 'weibullvariate']
>>>
```

Modules courants

— *math : fonctions et constantes mathématiques (sin, cos, exp, pi. . .).*

— *sys : passage d'arguments, gestion de l'entrée/sortie standard. . .*

— *os : dialogue avec le système d'exploitation (permet de sortir de Python, lancer une commande en shell. . .).*

— *random : génération de nombres aléatoires.*

— *time : accéder à l'heure de l'ordinateur et aux fonctions gérant le temps.*

— *profile : permet d'évaluer le temps d'exécution de chaque fonction dans un programme (profiling en anglais).*

— *urllib : permet de récupérer des données sur internet depuis Python.*

— *tkinter : (permet de créer des objets graphiques)*

— *pickle : écriture et lecture de structures Python (comme les dictionnaires par exemple).*

Les expressions régulières
regular expressions

Des expressions normalisées qui vous permettent de chercher et récupérer des informations dans un fichier

Une expression régulière est une suite de caractères qui a pour but de décrire un fragment de texte.
Elle est constitué de deux types de caractères :

— *Les caractères dits normaux.*
— *Les méta caractères ayant une signification particulière,*

^ ⇔ **Début de chaîne de caractères ou de ligne.**

Exemple : l'expression ^ATG correspond à la chaîne de caractères ATGCGT mais pas à la chaîne CCATGTT.

$ ⇔ **Fin de chaîne de caractères ou de ligne.**

Exemple : l'expression ATG$ correspond à la chaîne de caractères TGCATG mais pas avec la chaîne CCATGTT.

. ⇔ **N'importe quel caractère**

Exemple : l'expression A.G correspond à ATG, AtG, A4G, mais aussi à A-G ou à A G.

[ABC] ⇔**Le caractère A ou B ou C (un seul caractère).**

Exemple : l'expression T[ABC]G correspond à TAG, TBG ou TCG, mais pas à TG.

[A-Z] ⇔ **N'importe quelle lettre majuscule.**

Exemple : l'expression C[A-Z]T correspond à CAT, CBT, CCT. . .

[a-z] ⇔ **N'importe quelle lettre minuscule**

[0-9] ⇔**N'importe quel chiffre**

[A-Za-z0-9] ⇔ **N'importe quel caractère alphanumérique.**

[^AB] ⇔**N'importe quel caractère sauf A et B.**

Exemple : l'expression CG[^AB]T correspond à CG9T, CGCT. . mais pas à CGAT ni à CGBT.

**** ⇔ **Caractère d'échappement (pour protéger certains caractères).**

Exemple : l'expression \+ désigne le caractère + sans autre signification particulière. L'expression A\.G correspond à A.G et non pas à A suivi de n'importe quel caractère, suivi de G.

***** ⇔ **0 à n fois le caractère précédent ou l'expression entre parenthèses précédente.**

Exemple : l'expression A(CG)*T correspond à AT, ACGT, ACGCGT. . .

+ ⇔**1 à n fois le caractère précédent ou l'expression entre parenthèses précédente.**

Exemple : l'expression A(CG)+T correspond à ACGT, ACGCGT. . . mais pas à AT.

? ⇔**0 à 1 fois le caractère précédent ou l'expression entre parenthèses précédente.**

Exemple : l'expression A(CG)?T correspond à AT ou ACGT.

{n} ⇔**n fois le caractère précédent ou l'expression entre parenthèses précédente.**

Exemple : l'expression A(CG){2}T correspond à ACGCGT mais pas à ACGT, ACGCGCGT ou ACGCG.

{n,m} ⇔**n à m fois le caractère précédent ou l'expression entre parenthèses précédente.**

Exemple : l'expression A(C){2,4}T correspond à ACCT, ACCCT et ACCCCT mais pas à ACT, ACCCCCT ou ACCC.

{n,} ⇔ **Au moins n fois le caractère précédent ou l'expression entre parenthèses précédente.**

Exemple : l'expression A(C){2,}T correspond à ACCT, ACCCT et ACCCCT mais pas à ACT ou ACCC.

{,m} ⇔**Au plus m fois le caractère précédent ou l'expression entre parenthèses précédente.**

Exemple : l'expression A(C){,2}T correspond à AT, ACT et ACCT mais pas à ACCCT ou ACC.

(CG|TT) ⇔ **Les chaînes de caractères CG ou TT.**

Exemple : l'expression A(CG|TT)C correspond à ACGC ou ATTC.

Module re et la fonction search

Chercher un mot dans un texte est une tâche facile, c'est l'objectif de la méthode *find* attachée aux chaînes de caractères, elle suffit encore lorsqu'on cherche un mot au pluriel ou au singulier mais il faut l'appeler au moins deux fois pour chercher ces deux formes.

Pour des expressions plus compliquées, il est conseillé d'utiliser *les expressions régulières*. C'est une fonctionnalité qu'on retrouve dans beaucoup de langages. C'est une forme de grammaire qui permet de rechercher des expressions.

Dans le module **re**, la fonction **search()** permet de rechercher un motif (pattern) au sein d'une chaîne de caractères avec une syntaxe de la forme search (motif, chaine). Si motif existe dans chaine Python renvoie un objet du type SRE_Match

```
>>> import re                                    #appel du module   re
>>> animaux = "girafe tigre singe"
>>> re.search('tigre', animaux)                 #le module cherche la chaine
'tigre' dans la chaine animaux
<_sre.SRE_Match object at 0x7fefdaefe2a0> # ok c'est trouvé
>>> if re.search('tigre', animaux):                    #si oui dites "ok"
... print "OK"
...
OK
```

Remarquez l'absence de résultat de la condition

Questionnaire

1- Donnez la syntaxe générale d'ouverture et fermeture d'un fichier texte ?
2- Quelles sont les 3 modes d'ouverture?
3- Donnez la syntaxe générale de lecture d'un fichier
4- Donnez la syntaxe générale d'écriture dans un fichier
    a. Expliquez la différence entre les modes d'écriture "w" et "a"
5- Qu'est qu'un module?
6- Quelle est la syntaxe d'importation d'un module?
7- Définir les modules suivants (*math, os, random, time, urllib, tkinter, re)*
8- *Rappelez la syntaxe de la méthode find quelles sont ses limites?*
9- *Rappelez la syntaxe de recherche avec le module "re"*

Travaux pratiques

1. Créer un fichier texte nommé "test" avec un contenu *"salut je suis un test"*
2. Ecrire un script Python qui permet de:
    a. D'ouvrir le fichier en mode lecture dans une variable
    b. De lire le contenu du fichier dans une variable
    c. D'afficher le contenu du fichier
    d. De fermer le fichier

3. Ecrire un script Python qui permet de:
    a. D'ouvrir le fichier en mode écriture "w" dans une variable
    b. D'y écrire le texte suivant *" salut je suis un texte qui écrase le précédent "*
    c. D'ouvrir le fichier en lecture dans une variable
    d. D'afficher le contenu du fichier
    e. De fermer le fichier

4. Ecrire un script Python qui permet de rechercher le caractère "u" dans la chaine "salut je suis un test" en utilisant le model "re"

(solution page suivante)

Travaux pratiques

```
mon_fichier=open("test.txt","r")
contenu = mon_fichier.read()
print (contenu)
mon_fichier.close()
```

```
mon_fichier=open("test.txt","w")
mon_fichier.write("salut je suis un texte qui écrase le précédent")
mon_fichier=open("test.txt","r")
x=mon_fichier.read()
print (x)
mon_fichier.close()
```

```
import re
x="salut je suis un test "
y=re.search("u", x)
print (y)
```