

Découvrir Python sur la TI-83 Premium CE



unité
1

unité
2

unité
3

Auteur : Jean-Louis Balas



Sommaire

Unité 1 : Débuter la programmation en Python	1
Compétence 1 : Calculer avec Python.....	1
Compétence 2 : Les types de données en Python	5
Compétence 3 : Les fonctions en Python.....	8
Application : Les différents type de données Python.....	11
Unité 2 : Les boucles en programmation en Python	14
Compétence 1 : Instruction conditionnelle.....	14
Compétence 2 : La boucle bornée FOR	16
Compétence 3 : La boucle non bornée While.....	18
Application : Boucles et tests	21
Unité 3 : Exemples d'applications.....	23
Compétence 1 : Fonctions et boucles	23
Compétence 2 : La boucle bornée FOR	26
Compétence 3 : Programmation et récursivité.....	30
Application : Tests, boucles	33

Unité 1 : Débuter la programmation en Python

Compétence 1 : Calculer avec Python


Dans cette première leçon de l'unité 1, vous allez découvrir l'application TI-Python en utilisant les fonctions mathématiques les plus courantes incorporées à la calculatrice TI-83 Premium CE.

Objectifs :

- Utiliser le module TI-Python.
- Découvrir les fonctions mathématiques en Python.
- Distinguer l'éditeur de programmes et la console (Shell).
- Utiliser une instruction de programmation dans la console.

a) TI-83 Premium CE.

Connecter le module TI-Python à votre calculatrice en utilisant le câble mini USB. Respecter les connexions (Brochage B sur le module)

Lorsque votre module Python est connecté à la calculatrice, un indicateur (carré vert) le précise juste à côté de celui de la charge de la batterie. 

b) TI-83 Premium CE Edition Python.

Appuyer sur la touche **prgm**
Choisir l'application **Python App**

Remarque : L'application « **Python** » est également accessible à partir des applications de la calculatrice **2nde apps**



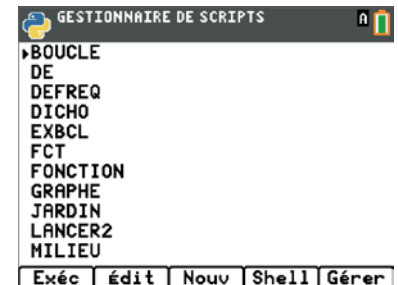
Conseil à l'enseignant : L'utilisation de l'application TI-Python nécessite la mise à jour de l'OS de la calculatrice vers la version 5.35 ou supérieure. Il est également recommandé de mettre à jour le logiciel TI-Connect™ CE afin de pouvoir directement transférer des programmes conçus en Python sur un IDLE (Environnement intégré de développement) directement vers la calculatrice TI-83 Premium CE.

Mettre la calculatrice sous tension et appeler l'application **PyAdaptr**

Valider en appuyant sur la touche **enter**

Vous devez également voir la diode verte du module TI-Python allumée.

Les touches $F_1 \dots F_5$ sont utilisées pour accéder à toutes les fonctionnalités de l'éditeur.



Conseil à l'enseignant : L'utilisation du langage Python s'effectue généralement à partir d'un script que l'on exécute dans la console. Cependant, dans la console, il est possible de :

- Faire des calculs, définir des variables afin de les intégrer dans des calculs.
- Écrire et exécuter un programme.
- Exécuter un programme saisi dans l'éditeur et demander les valeurs prises par les variables de ce programme.

Dans un premier temps nous allons utiliser la console également appelée « Shell »

Appuyer sur la touche **F4** pour accéder à la console (Shell)

Quelques commandes de base.

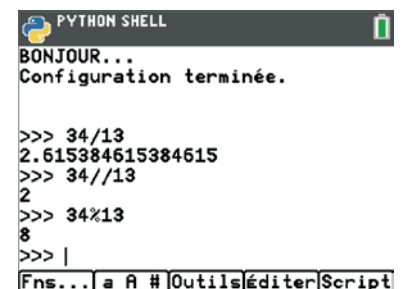
Les variables sont généralement nommées par des lettres minuscules. Pour y

accéder ainsi qu'aux commandes de base, appuyer sur **F2** $\left[\begin{smallmatrix} a \\ \# \end{smallmatrix} \right]$

$c \leftarrow 5$ va s'écrire en Python $c = 5$ et s'obtient sur la calculatrice en tapant :

(c $\left[\begin{smallmatrix} \text{sto} \rightarrow \\ 5 \end{smallmatrix} \right]$). **Cette instruction signifie que 5 est affecté à la variable c.**

Pour tester la valeur de la variable c : on écrira $c == 5$ ou bien $c >= 5 \dots$



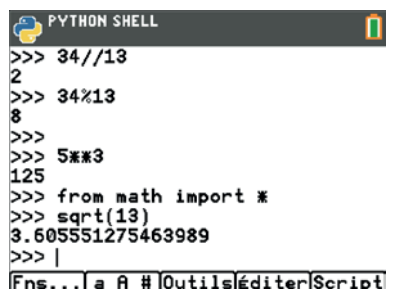
Les calculs classiques :

- Le reste de la division de a par b s'écrit $a\%b$
- Le quotient euclidien de a par b s'écrit $a//b$
- x à la puissance n s'écrit $x**n$. On peut aussi écrire $\text{pow}(x,n)$

Remarque : Le chargement du module (bibliothèque) « **math import** » est nécessaire pour effectuer des calculs sur les racines carrées et sur les fractions.

Pour incorporer ce module, appuyer sur la touche **F1** $\left[\begin{smallmatrix} \text{Fns} \dots \\ \# \end{smallmatrix} \right]$ et choisir **Modul** puis enfin le menu **1 : math...**

- La racine carrée de x ($x \geq 0$) s'écrit **sqrt(x)**
- Le nombre π s'écrit **pi**



Conseil à l'enseignant : Les touches de la calculatrice TI-83 Premium CE fonctionnent comme habituellement. Par exemple, l'appui sur la touche **[math]** permet ainsi de choisir l'insertion dans la console ou dans un script du module **math** ou **random**.

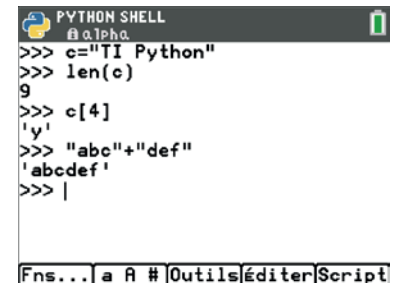
Il est également possible d'y accéder en appuyant sur la touche **F1** $\left[\begin{smallmatrix} \text{Fns} \dots \\ \# \end{smallmatrix} \right]$ qui désigne l'ensemble des fonctions Python.

Les commandes relatives aux chaînes de caractères.

Les chaînes de caractères se définissent à l'aide de guillemets doubles ou simples.

« TI-Python » ou bien 'TI-Python'

- Obtenir la longueur d'une chaîne de caractères **len(c)** (Menu **Fns...** puis **List**)
- `c[k]` renvoie le k+1 élément de la chaîne c.
- Pour concaténer deux chaînes de caractères, simplement les additionner.



```

PYTHON SHELL
@a,1pha
>>> c="TI Python"
>>> len(c)
9
>>> c[4]
'y'
>>> "abc"+"def"
'abcdef'
>>> |
Fns...|a A #|Outils|Éditer|Script
    
```

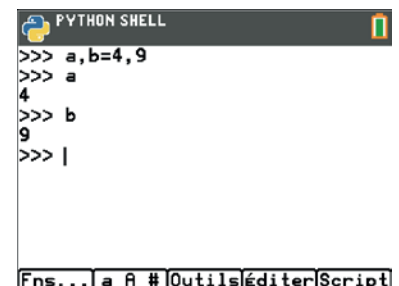
Conseil à l'enseignant : l'appui simple sur la touche `[alpha]` permet de configurer le clavier en minuscules.

L'appui double donne accès aux majuscules. La séquence de touches `[2nde][alpha]` permet de bloquer le clavier en mode alpha numérique et un nouvel appui sur `[alpha]` change le clavier d'un mode à l'autre.

Remarque : Pour effacer une console des événements précédents, appuyer sur **F3** (Outils) et choisir le menu **5 : Effacer l'écran** ou **6 : Nouveau Shell** si vous ne souhaitez pas conserver les variables.

Un peu d'astuce :

Lors de l'affectation de plusieurs variables, il est possible de le faire en une seule fois comme le montre l'écran ci-contre.



```

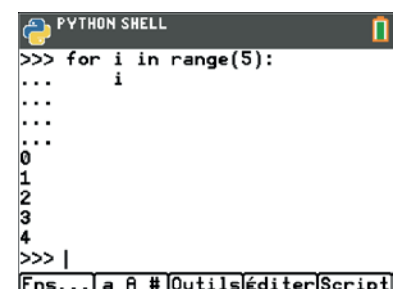
PYTHON SHELL
>>> a,b=4,9
>>> a
4
>>> b
9
>>> |
Fns...|a A #|Outils|Éditer|Script
    
```

Utiliser une instruction de programmation dans le Shell.

Le langage Python possède la richesse de pouvoir observer indépendamment d'un script, une fonctionnalité particulière.

Ainsi sur l'écran de droite, on peut analyser le fonctionnement d'une boucle **for** à laquelle on accède en appuyant sur **F1** puis en choisissant dans le menu **Ctl** l'option **5 : for i in range(début , fin)** :

Nous reviendrons sur les boucles dans une leçon ultérieure.



```

PYTHON SHELL
>>> for i in range(5):
...     i
...
...
...
0
1
2
3
4
>>> |
Fns...|a A #|Outils|Éditer|Script
    
```

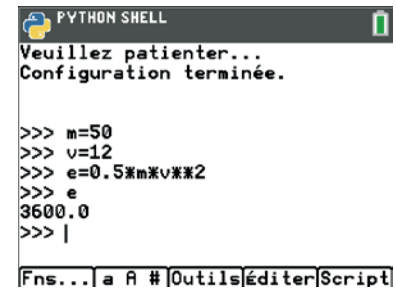
Appliquons nos connaissances.

L'énergie cinétique d'un solide en mouvement est donnée par la relation $E_c = \frac{1}{2}mv^2$

m est la masse du solide en kg

v est sa vitesse en m/s

En utilisant la console, quelle est la valeur de la variable *energie* si la *masse* est de 50 kg et la *vitesse* de 12 m/s ?



```
PYTHON SHELL
Veillez patienter...
Configuration terminée.

>>> m=50
>>> v=12
>>> e=0.5*m*v**2
>>> e
3600.0
>>> |

Fns... | a A # |Outils|Éditer|Script
```

Conseil à l'enseignant : Un programme informatique contient des instructions qui utilisent des variables. Une variable est une « case » qui permet de conserver des données du programme (nombre, valeur entrée par l'utilisateur, chaîne de caractères ...) en les stockant dans la mémoire de l'ordinateur. L'affectation d'une valeur dans une variable se fait à l'aide de la touche **[sto→]** qui recopie dans le Shell le signe **=**

A partir de la console (Shell), l'accès au catalogue **[2nde] [0]** permet d'accéder à toutes les fonctionnalités en Python disponibles dans la calculatrice.

Pour quitter l'application Python, procéder comme avec toutes les applications en appuyant sur les touches **[2nde] [mode]** puis **F5 (OK)**

Unité 1 : Débuter la programmation en Python

Compétence 2 : Les types de données en Python

Dans cette deuxième leçon de l'unité 1, vous allez découvrir comment utiliser le type des données en Python.

Objectifs :

- Connaître les différents types de données en langage Python.
- Mettre en forme le format d'une donnée numérique.


Connaître le type des grandeurs utilisé.

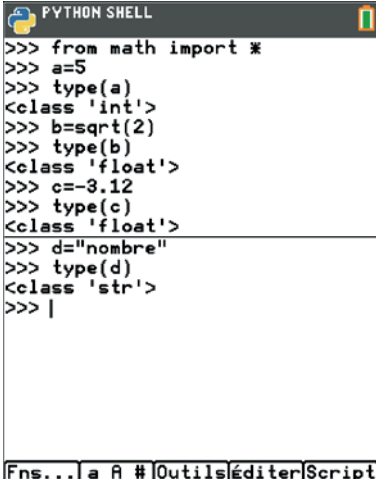
Lorsque vous utilisez un script en langage Python, il peut s'avérer nécessaire de connaître le type de variable utilisé, ou bien de modifier ces variables en vue d'une utilisation ultérieure. Par exemple si la grandeur renvoyée par un script Python renvoie une grandeur correspondant à une mesure en sciences physiques, il n'est pas nécessairement opportun de conserver un résultat à 6 décimales.

Vous allez créer un script dans l'application **Python** permettant de distinguer les types des grandeurs utilisées.

- Une chaîne de caractère
- Un nombre réel
- Un nombre irrationnel, $\sqrt{2}$ par exemple


Vous afficherez le nombre ainsi que son type à l'aide l'instruction **type**.

- Importer le module **maths** (Fns... puis  Modul puis 1 : math... et enfin 1 : from math import *) puis appuyer sur la touche **entrer**.
- L'ensemble des commandes utilisé peut être obtenu en appuyant sur la touche **F1** Fns... puis en choisissant les sous menus **E/S** (entrée/sortie) et **Type**
- La commande **type** qui permet de connaître la nature d'une variable est tapée à la main.



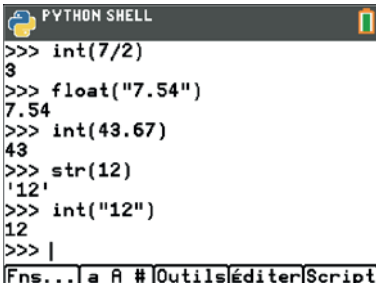
```

PYTHON SHELL
>>> from math import *
>>> a=5
>>> type(a)
<class 'int'>
>>> b=sqrt(2)
>>> type(b)
<class 'float'>
>>> c=-3.12
>>> type(c)
<class 'float'>
>>> d="nombre"
>>> type(d)
<class 'str'>
>>> |
    
```

Astuce : L'utilisation des touches de direction  permet de recopier une ligne lorsque l'utilisateur travaille dans la console.

Remarques :

- L'opérateur **int()** extrait lorsque c'est possible, un entier d'une chaîne de caractères et renvoie la partie entière d'un nombre comprise entre $\pm 2\ 147\ 483\ 648$ (codage sur 32 bits, soit 4 octets)
- L'opérateur **str()** transforme un nombre en une chaîne de caractères.
- L'opérateur **float()** extrait lorsque c'est possible, un flottant d'une chaîne de caractères.



```

PYTHON SHELL
>>> int(7/2)
3
>>> float("7.54")
7.54
>>> int(43.67)
43
>>> str(12)
'12'
>>> int("12")
12
>>> |
    
```

Une autre astuce :

Pour incrémenter une variable dans un compteur, on dispose de deux possibilités :

a) Écrire par exemple : `compteur = 0`

demander l'affichage de la variable

b) Ou bien

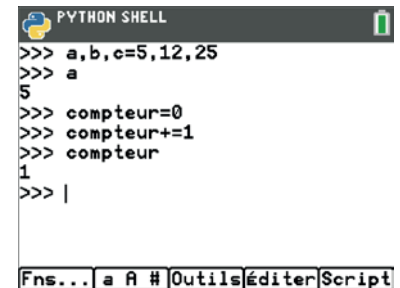
`compteur = compteur + 1`

`compteur`

`compteur = 0`

`compteur+=1`

`compteur`



```
PYTHON SHELL
>>> a,b,c=5,12,25
>>> a
5
>>> compteur=0
>>> compteur+=1
>>> compteur
1
>>> |
Fns... | a A # |Outils|Éditer|Script
```

Conseils à l'enseignant : Penser à utiliser le clavier alphanumérique `[2nde]` `[alpha]` et `[alpha]` pour passer du mode majuscule à minuscule.

Le symbole « `_` » est obtenu en utilisant la palette a A # accessible par la touche **F2**.

Utiliser les fonctions de « copier-coller » à partir du menu **F3 Outils**

Lors de la rédaction, la touche `[suppr]` permet d'effacer un caractère entré par erreur

A partir de la console (Shell), l'accès au catalogue `[2nde]` `[0]` permet d'accéder à toutes les fonctionnalités en Python disponibles dans la calculatrice.

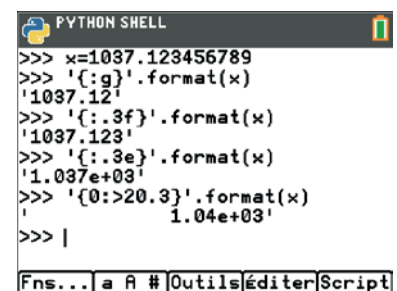
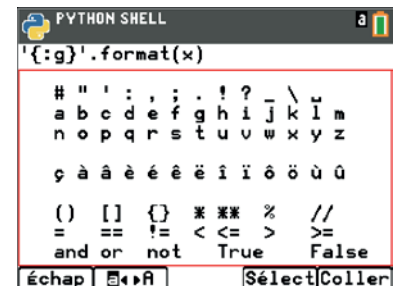
Vous avez la possibilité de commenter vos scripts en ajoutant devant le commentaire, un (`#` commentaire). Le fait de mettre un `#` indique que la ligne ne sera pas interprétée. Pour l'obtenir, appuyer sur la touche **F2** (a A #)

Pour aller plus loin :

Le format des nombres : La méthode format de l'objet string est un outil très puissant permettant de créer des chaînes de caractères en remplaçant certains champs (entre accolades) par des valeurs (passées en argument de la fonction format) après conversion de celles-ci. On peut préciser à l'intérieur de chaque accolade un code de conversion, ainsi que le gabarit d'affichage. Donnons quelques exemples.

```
>>> x = 1037.123456789
>>> '{:g}'.format(x) # choisit le format le plus approprié '1.04e+03'
>>> '{:.3f}'.format(x) # fixe le nombre de décimales
'1037.123'
>>> '{:.3e}'.format(x) # notation scientifique
'1.037e+03'
>>> '{0 :20.3f}'.format(x) # précise la longueur de la chaîne
' 1037.123'
>>> '{0 :>20.3f}'.format(x) # justifié à droite
' 1037.123'
>>> '{0 :<20.3f}'.format(x) # justifié à gauche
'1037.123 '
>>> '{0 :^20.3f}'.format(x) # centré
' 1037.123 '
>>> '{0 :+.3f} ; {1 :+.3f}'.format(x, -x) # affiche toujours le signe '+1037.123 ; -1037.123'
'1037.123 ; -1037.123'
>>> '{0 :.3f} ; {1 :.3f}'.format(x, -x) # affiche un espace si x>0
'1037.123 ; -1037.123'
```

Appuyer sur la touche **F2 (a A #)** afin d'accéder à la table de caractères



Unité 1 : Débuter la programmation en Python

Compétence 3 : Les fonctions en Python

Dans cette troisième leçon de l'unité 1, vous allez utiliser l'éditeur de programme (script) afin de créer des fonctions, puis exécuter celui-ci afin d'observer les résultats dans la console.

Objectifs :

- Découvrir la notion de fonction en Python.
- Créer une fonction.

Vers la notion de fonction en Python.

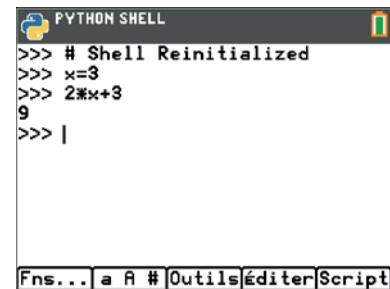
Mettre en œuvre l'algorithme suivant :

$$x \leftarrow 3$$
$$y \leftarrow 2 \times x + 3$$

Dans une console, ceci se réalise très simplement.

Mais si l'on souhaite répéter ce type de calcul pour une autre valeur de x il faut écrire de nouveau l'ensemble de la séquence. Ce qui, sur un exemple moins trivial, peut s'avérer vite fastidieux.

On est donc conduit à créer une fonction qui nous permettra de dupliquer aisément le traitement de l'algorithme.



```
PYTHON SHELL
>>> # Shell Reinitialized
>>> x=3
>>> 2*x+3
9
>>> |
```

En algorithmique, une fonction peut être considérée comme une séquence d'instructions, réalisant une certaine tâche, en utilisant un ou plusieurs **arguments**.

Cette fonction reçoit un nom.

- La programmation d'une fonction commence toujours par **def** suivi du nom de la fonction, suivi des arguments de celle-ci. Cette ligne se termine par le symbole **:**
- Les deux points marquent le démarrage du bloc d'instructions définissant la fonction : toutes ces instructions sont **indentées**, c'est-à-dire décalées vers la droite par rapport à la première ligne. On ajoute en tête de chaque ligne, le même nombre d'espaces

La fonction renvoie un seul résultat par l'intermédiaire de la commande **return**. Le résultat peut être constitué d'une liste de résultats, une chaîne de caractères...

def nom_fonction(liste des arguments) :

- .. bloc d'instructions
- .. return (résultats)

L'indentation, obtenue avec la touche de tabulation ou avec des espaces, est **primordiale** : tout ce qui est indenté après le **def()** sera exécuté comme un bloc. Il ne faut pas que l'indentation varie (nombre d'espaces, passer de la tabulation à des espaces. . .) en cours de bloc.

Conseil à l'enseignant : Une fonction permet de découper le problème étudié en sous problèmes et d'éviter ainsi la répétition d'instructions. Une fois définie, elle peut être « appelée » tout au long de l'exécution du programme autant de fois que nécessaire.

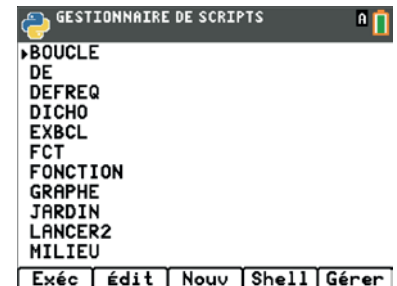


Une fonction peut n'avoir aucun argument. Elle peut également être appelée dans un autre programme : il suffit pour cela de l'insérer dans une instruction en saisissant son nom et les valeurs des arguments.

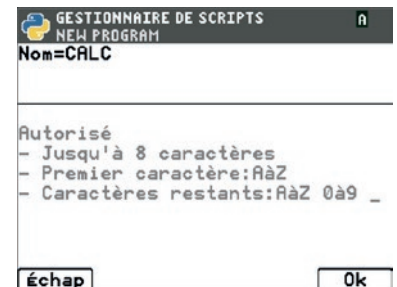
Mise en œuvre d'un premier exemple.

- Créer un nouveau script

Ouvrir l'application **PyAdaptr** et choisir la création d'un nouveau script en appuyant sur **F3** « Nouv »



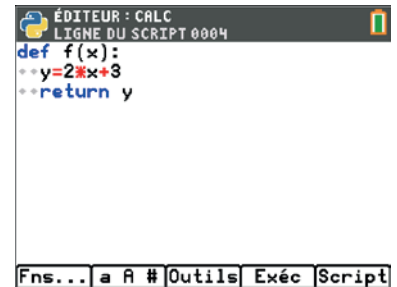
- Donner un nom à votre script. Par exemple CALC puis valider en appuyant sur **Entrer** (Le nombre maximal de caractères est fixé à 8)
- Le nom de votre script apparaît ensuite en haut dans le bandeau gris **EDITEUR : CALC**



- Choisir ensuite l'onglet **F1** Fns... et utiliser les touches de direction **→** afin d'atteindre le menu **Fonc**
- Choisir le menu **1** : **def fonction()** : la parenthèse ouvrante clignote. Appuyer sur **alpha** afin de donner un titre à la fonction, mettre le paramètre **u** à l'intérieur des parenthèses puis valider par **Entrer**



- Vous devriez obtenir l'écran ci-contre. Vous remarquerez l'indentation automatique du curseur
- Continuez ensuite par les instructions de traitement de l'algorithme. Souvenez-vous que l'affectation d'une variable s'effectue en appuyant sur la touche `sto→`
- Terminer votre script en appuyant de nouveau sur **F1** Fns... puis les touches de direction `▸` afin d'atteindre le menu **Fonc**
- Choisir seconde option 2 : **return** puis compléter l'instruction.

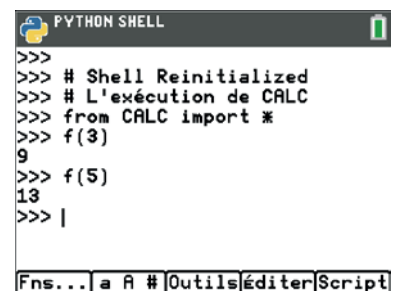


```
ÉDITEUR : CALC
LIGNE DU SCRIPT 0004
def f(x):
  y=2*x+3
  return y

Fns... | a A # | Outils | Exéc | Script
```

A présent vous êtes prêts pour exécuter votre script.

- Appuyer sur l'onglet **F4** Exec
- La console s'affiche et un message vous informe du chargement du script
- Compléter l'invite de commande par le nom de la fonction avec les arguments prévus (un seul dans cet exemple), puis valider.
- Les touches de déplacement `▲` `▼` permettent de rappeler la dernière Entrée (voir la section **Outils**)



```
PYTHON SHELL
>>>
>>> # Shell Reinitialized
>>> # L'exécution de CALC
>>> from CALC import *
>>> f(3)
9
>>> f(5)
13
>>> |

Fns... | a A # | Outils | éditer | Script
```

Unité 1 : Débuter la programmation en Python

Application : Les différents type de données Python

Écrire quelques scripts permettant de réinvestir les notions vues dans les leçons de l'unité 1

- Fonction en langage Python
- Création d'une liste

Objectifs :

- Créer un convertisseur de température.
- Créer un script permettant de développer une expression algébrique.

Exemple n°1 : Convertir une température.


Pour mesurer la température en France, on utilise le degré Celsius (°C). Dans les pays anglo-saxons, on utilise le degré Fahrenheit (°F).

Votre travail consiste à programmer une fonction qui réalise la conversion de température dans les deux sens : °C ↔ °F

Existe-t-il une température qui soit égale dans les deux unités ?

On rappelle : $t(^{\circ}F) = \frac{9}{5} \times t(^{\circ}C) + 32$ et en première approximation on peut utiliser $t(^{\circ}F) = t(^{\circ}C) \times 1.8 + 32$



- Ouvrir l'application Python et commencer un nouveau script **F3** (Nouv)
- Nommer le script TEMPERAT et valider en appuyant sur **F5** (OK)
- Importer le module **maths** (Fns... puis  **Modul** puis **1 : math...** et enfin **1 : from math import ***)
- Créer une première fonction de conversion °C → °F (Fns... puis ~ Fonc et enfin def **fonction() : :**)
- Exécuter le début du script en réalisant la conversion en degré Fahrenheit d'une température de 40°C f(40).

```
ÉDITEUR : TEMPERAT
LIGNE DE SCRIPT 0004
from math import *
def f(c):
    F=1.8*c+32
    return F
```

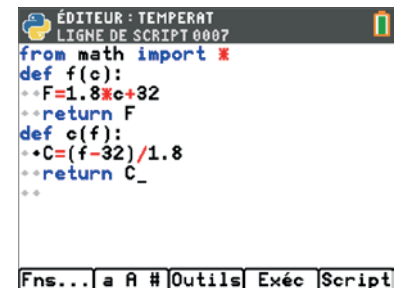
Fns... | a A # | Outils | Exéc | Script

```
PYTHON SHELL
>>>
>>> # Shell Reinitialized
>>> # L'exécution de TEMPERAT
>>> from TEMPERAT import *
>>> f(40)
104.0
>>> |
```

Fns... | a A # | Outils | Éditer | Script

Conseil à l'enseignant : Lors de l'exécution d'un script (mode console (Shell)), l'appui sur la touche **VAR** permet d'appeler la fonction sans arguments. Pour l'utiliser, appuyer sur **F5** (OK) puis compléter la fonction avec les arguments attendus.

- Terminer la réalisation du script en rajoutant à la suite les instructions nécessaires à la conversion $^{\circ}F \rightarrow ^{\circ}C$.
- Remarque utile : Utiliser dans la palette Outils l'option **2 : Indent**← afin de revenir à la ligne non indentée (sinon, un message d'erreur s'affichera lors de l'exécution du script).
- En résumé, la fonction $f(c)$ donne une température $^{\circ}C \rightarrow ^{\circ}F$ et la fonction $c(f)$ réalise la conversion $^{\circ}F \rightarrow ^{\circ}C$



```

ÉDITEUR : TEMPERAT
LIGNE DE SCRIPT 0007
from math import *
def f(c):
    F=1.8*c+32
    return F
def c(f):
    C=(f-32)/1.8
    return C

```

Pour rechercher enfin une valeur de la température qui soit identique dans les deux unités, de nombreuses méthodes sont possibles mettant en œuvre les boucles et les tests que nous verrons dans les unités 2 et 3.

Nous allons procéder en balayant une boucle avec un pas de 10° (à affiner éventuellement, et créer dans un autre script.



GESTIONNAIRE DE SCRIPTS

- ▶ TAB
- TEMP2
- TEMPERAT

Exéc | Éditer | Nouv | Shell | Gérer

Aller dans le gestionnaire de scripts en appuyant sur **F5**

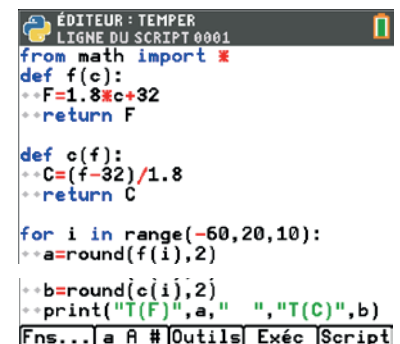
- Mettre le curseur face au script TEMPERAT
- Choisir l'onglet **F5 (Gérer)** puis choisir **1 : Dupliquer le script**
- Donner un autre nom au script (TEMP2) par exemple, puis valider par **F5** OK.

Le script est ainsi dupliqué sous un autre nom.

Nous rechercherons la solution dans l'intervalle $[-60 ; 10]$ par pas fixé à 10° dans un premier temps. Le pas est une donnée à demander à l'utilisateur

Nous allons utiliser trois instructions supplémentaires

- `round(a,2)` afin d'arrondir un nombre « a » à 2 décimales
- `for i in range(début , fin , pas)` afin de créer une double liste
- Et enfin l'instruction **print** afin d'afficher les résultats



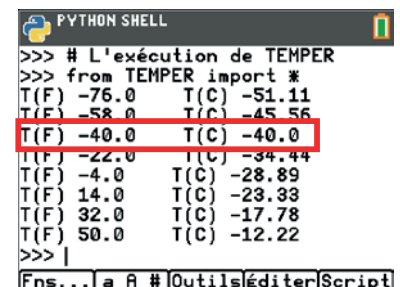
```

ÉDITEUR : TEMPER
LIGNE DU SCRIPT 0001
from math import *
def f(c):
    F=1.8*c+32
    return F
def c(f):
    C=(f-32)/1.8
    return C
for i in range(-60,20,10):
    a=round(f(i),2)
    b=round(c(i),2)
    print("T(F)",a," ", "T(C)",b)

```

Les fonctions s'obtiennent par **F1 (Fns...)** puis à partir des menus E/S ; Type et **Ops**. L'instruction `round()` se trouve dans le catalogue, mais peut aussi être entrée manuellement.

Vous devriez obtenir les résultats de droite après avoir exécuté le script **F4**



```

PYTHON SHELL
>>> # L'exécution de TEMPER
>>> from TEMPER import *
T(F) -76.0 T(C) -51.11
T(F) -58.0 T(C) -45.56
T(F) -40.0 T(C) -40.0
T(F) -22.0 T(C) -34.44
T(F) -4.0 T(C) -28.89
T(F) 14.0 T(C) -23.33
T(F) 32.0 T(C) -17.78
T(F) 50.0 T(C) -12.22
>>> |

```

Conseil à l'enseignant : Pour affiner le script, on pourra éventuellement à le modifier en incitant l'élève à proposer un intervalle de variation à faire fixer par l'utilisateur, ainsi que la valeur du pas. On utilisera donc une instruction du type `p=float(input(« Pas= »))`.

Remarque importante : Attention lors de l'exécution d'une boucle de type `for i in range(début , fin , pas)` (le range s'arrête à « fin moins une valeur du pas »)

La boucle FOR, sera abordée lors de l'étude de l'unité 2 compétence 2.

Unité 2 : Les boucles en programmation en Python

Compétence 1 : Instruction conditionnelle

Dans cette première leçon de l'unité 2, vous allez découvrir comment écrire et utiliser une instruction conditionnelle en Python.

Objectifs :

- Écrire et utiliser une instruction conditionnelle.
- Réinvestir la notion de fonction en Python.

Dans un programme, il est particulièrement fréquent d'avoir à orienter l'exécution de celui-ci en fonction de **conditions** qui affectent les différentes variables.

Une condition est un énoncé qui peut être **vrai** ou **faux**.

Par exemple : $a = b$ ou bien $a \geq b$ mais aussi n est pair sont des conditions qui sont vérifiées selon les valeurs affectées à ces variables.

Dans un programme, on peut tester une condition et selon que celle-ci est vraie ou fausse, effectuer un traitement ou un autre. On parle alors de **traitement conditionnel**.

```
if condition :
    Instruction A
else :
    Instruction B
```

Conseil à l'enseignant : En langage Python, il n'y a pas d'instruction pour indiquer la fin de l'instruction conditionnelle. C'est l'indentation qui décale vers la droite les instructions A et B.

`elif` est la contraction de `else if`

Pour tester l'égalité de deux valeurs en langage Python, on utilise le signe « == »

Exemple :

Une société de location de voitures propose à ses clients le contrat suivant :

Un forfait de 66 € auquel s'ajoute 0.25 € par kilomètre au-delà de 70 km.

Votre travail consiste à écrire un script qui permette de calculer automatiquement le coût C du contrat en fonction de la distance parcourue.

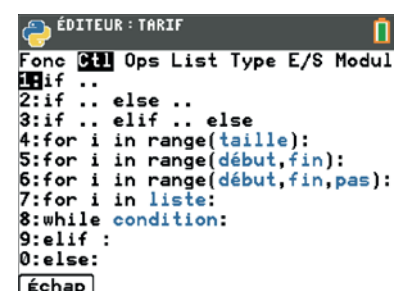
Langage naturel

$X \leftarrow a$
Si ($0 < X$) et ($X < 70$)
Alors C prend la valeur 66
Sinon C prend la valeur $66 + 0.25X$
Fin Si

Conseil à l'enseignant : Prévoir éventuellement le cas où l'utilisateur saisit un nombre X négatif

Mise en œuvre :

- Démarrer l'application (**PyAdaptr pour TI-83 Premium CE & adaptateur** ou **Python App pour TI-83 Premium CE Edition Python**)
- Commencer un nouveau script **F3** (Nouv) et le nommer « TARIF ». Valider en appuyant sur la touche **F5** (OK)
- Appuyer sur la touche **F1** (Fns..)
- L'instruction **If** est obtenue à partir du menu **Ctl**
- Choisir le menu **3:if..elif..else**



```
ÉDITEUR : TARIF
Fonc Ctrl Ops List Type E/S Modul
1:if ..
2:if .. else ..
3:if .. elif .. else
4:for i in range(taille):
5:for i in range(début,fin):
6:for i in range(début,fin,pas):
7:for i in liste:
8:while condition:
9:elif :
0:else:
Échap
```


- Observer l'indentation du bloc conditionnel
- Le compléter en utilisant l'algorithme proposé en langage naturel

```
ÉDITEUR : TARIF
LIGNE DU SCRIPT 0002
def c(x):
    *if :
    *
    *elif :
    *
    *else:
    *
    *return c
```

- Vous devriez obtenir le script ci-contre.
- L'instruction `<` et `<=` ainsi que `and` se trouve dans le menu (Fns...) puis **Ops** mais sont également accessibles en appuyant sur les touches `2nde` `math`.

```
ÉDITEUR : TARIF
LIGNE DU SCRIPT 0008
def c(x):
    *if 0<x and x<70:
    *c=70
    *elif x<=0:
    *c=0
    *else:
    *c=66+(x-70)*0.25
    *return c
```

- Exécuter le script pour déterminer le coût d'un trajet avec une voiture louée à cette société.

```
PYTHON SHELL
Configuration terminée.

>>> # Shell Reinitialized
>>> # L'exécution de TARIF
>>> from TARIF import *
>>> c(70)
66.0
>>> c(123)
79.25
>>> |
```

Appliquons nos connaissances : Fonction par morceaux

On donne la fonction affine par morceaux f définie par :

$$f(x) = \begin{cases} 2x + 1 & \text{si } x \leq -1 \\ -x + 2 & \text{si } x \in] -1 ; 0] \\ -3x + 2 & \text{si } x > 0 \end{cases}$$

Copier le script ci-contre et l'exécuter afin de compléter le tableau ci-dessous

x	-4	-1.5	-0.5	-0.1	0.6	2.5	4.8	7.3
$f(x)$								

```
ÉDITEUR : FCT
LIGNE DU SCRIPT 0001
def f(x):
    *if x<=-1:
    *f=2*x+1
    *elif x>-1 and x<=0:
    *f=-x+2
    *else:
    *f=-3*x+2
    *return f
```

```
PYTHON SHELL
Configuration terminée.

>>> # Shell Reinitialized
>>> # L'exécution de FCT
>>> from FCT import *
>>> f(-5)
-9
>>> f(-4)
-7
>>> |
```

Unité 2 : Les boucles en programmation en Python

Compétence 2 : La boucle bornée FOR

Dans cette seconde leçon de l'unité 2, vous allez découvrir comment répéter un processus ou un ensemble d'instructions en utilisant une boucle bornée **FOR**.

Objectifs :

- Découvrir et mettre en oeuvre la boucle bornée **FOR**.
- Utiliser la boucle **FOR** dans des exemples simples.

Il est parfois utile dans un programme de répéter une ou plusieurs instructions un nombre défini de fois. Si le nombre de répétition du processus est connu à l'avance, on utilise une boucle bornée **For**.

La syntaxe d'une boucle For est la suivante :

Langage naturel

Pour variable **allant de** minimum à **maximum**
Instructions

Langage Python

for variable in **range ()** :
Instructions

La fonction **range()** permet d'énumérer le nombre de passages dans la boucle bornée. Elle peut être appelée de plusieurs façons :

- **for i in range(taille)** : prend les valeurs entières de 0 à *taille* - 1, donc "*taille*" valeurs.
- **for i in range(début, fin)** : la variable *i* prend des valeurs entières de *début* à *fin* - 1, où *début* et *fin* sont ici des entiers.
- **for i in range(début, fin, pas)** : la variable *i* prend des valeurs entières de *début* à *fin* - 1 par valeurs s'incrémentant de *pas*. *début*, *fin* et *pas* sont ici des entiers.
- **for i in liste** : la variable *i* utilisera directement les valeurs de la liste de la première valeur jusqu'à la dernière.

Il n'existe pas d'instruction de fin de boucle. C'est l'indentation, c'est-à-dire le décalage vers la droite d'une ou plusieurs lignes, qui permet de marquer la fin de la boucle.

Mise en œuvre :

Vous allez créer un script permettant de bien comprendre ce qu'est une boucle ainsi qu'un processus d'itération.

- Commencer un nouveau script et le nommer « BOUCLE »
- Initialiser une liste vide avec l'instruction **b=[]**. En langage Python, les éléments d'une listes sont placés entre **[]**, séparés par des virgules.
- Appuyer sur **F1** (Fns...) et choisir dans le menu Ctl, l'option **4 : for i in range(taille)**
- L'instruction **.append()** permet de compléter une liste. Ainsi, **b.append(i**2)** incrémente la liste **b** des carrés de *i*. A chaque valeur de *i*, la valeur de *i*² est placée en fin de liste
- La variable *i* variant de 0 à 4 ce qui correspond bien à un éventail de 5 valeurs.

Conseil à l'enseignant : Attention, les compteurs de boucles sont toujours initialisés à 0.

L'instruction `.append` concerne les listes. Pour l'atteindre, appuyer sur **F1 (Fns...)**, puis choisir le menu **List** et enfin choisir **6 : .append(x)**

- Appuyer sur la touche **F4** pour exécuter le script.

Demander ensuite l'affichage des valeurs de **b** à partir de la console

```

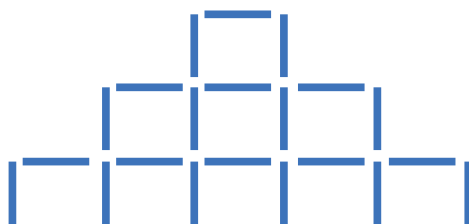
PYTHON SHELL
>>>
>>> # Shell Reinitialized
>>> # L'exécution de BCL
>>> from BCL import *
>>> b
[0, 1, 4, 9, 16]
>>> |
    
```

Conseil à l'enseignant : Dans une boucle ou une instruction comportant une indentation, toute écriture indentée d'une commande fait partie de la boucle. La fin de la boucle est marquée par la sortie de l'indentation.

Appliquons nos connaissances :

On réalise une construction à base de bâtons.

La première rangée notée « rangée 0 » est formée de 3 bâtons, la seconde de 7 bâtons et la troisième rangée de 11 bâtons.



Rangée n°0

Rangée n°1

Rangée n°2

De combien de bâtons sera formée la rangée n°4.

Réaliser le script, puis l'exécuter

- Exécuter le programme pour déterminer combien la 100^{ème} rangée comptera de bâtons.

```

ÉDITEUR : CRAYON
LIGNE DU SCRIPT 0001
def c(n):-
    x=3
    for i in range(1,n+1):
        x=x+4
    return x
    
```

```

PYTHON SHELL
>>>
>>> # Shell Reinitialized
>>> # L'exécution de CRAYON
>>> from CRAYON import *
>>> c(99)
399
>>> |
    
```

Conseil à l'enseignant : Appuyer sur la touche **VAR** lors de l'exécution d'un script afin de rappeler les variables ou fonctions écrites dans celui-ci.

Unité 2 : Les boucles en programmation en Python

Compétence 3 : La boucle non bornée While

Dans cette troisième leçon de l'unité 2, vous allez découvrir comment répéter un processus ou un ensemble d'instructions en utilisant une boucle non bornée **While**.

Objectifs :

- Découvrir et mettre en oeuvre la boucle non bornée **While**.
- Utiliser la boucle **While** dans des exemples simples.

Il est parfois utile dans un programme de répéter une ou plusieurs instructions un nombre indéfini de fois. Si le nombre de répétition du processus n'est pas connu à l'avance, on utilise une boucle non bornée **While**.

La boucle est alors parcourue un nombre de fois jusqu'à ce qu'une condition ne soit plus vérifiée. Tant que cette condition est vérifiée, la boucle continue.

La syntaxe d'une boucle **While** est la suivante :

Langage naturel

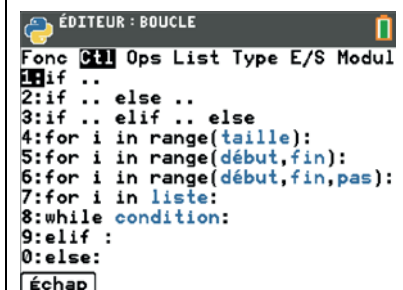
Tant que condition faire

Instructions

Langage Python

`while` condition :

Instructions



```

ÉDITEUR : BOUCLE
Fonc Ctrl Ops List Type E/S Modul
1:if ..
2:if .. else ..
3:if .. elif .. else
4:for i in range(taille):
5:for i in range(début,fin):
6:for i in range(début,fin,pas):
7:for i in liste:
8:while condition:
9:elif :
0:else:
Échap
    
```

Il n'existe pas d'instruction de fin de boucle. C'est l'indentation, c'est-à-dire le décalage vers la droite d'une ou plusieurs lignes, qui permet de marquer la fin de la boucle.

Mise en œuvre : Exemple n°1

Vous allez créer un script permettant de bien comprendre ce qu'est une boucle ainsi qu'un processus d'itération.

Mettre en œuvre l'algorithme ci-dessous :

L'algorithme a pour objet de déterminer le plus petit entier n tel que le terme général de la suite définie par $c_0 = 3,4$ et $c_{n+1} = 0,8 c_n$ est inférieur à 1.

$n \leftarrow 0$
 $c \leftarrow 3.4$
 Tant que $c \geq 1$
 $n \leftarrow n + 1$
 $c \leftarrow 0.8c$
 Fin Tant que

- Commencer un nouveau script et le nommer « WHILE »
- L'instruction **While** est accessible en tapant sur **F1** (Fns...) puis en choisissant le menu **Ctrl** et enfin l'instruction **8 : while condition :**
- Tant que la variable c sera strictement supérieure au seuil la variable n sera incrémentée de 1.



```

ÉDITEUR : WHILE
LIGNE DU SCRIPT 0001
def a(c,seuil):_
  n=0
  while c>=seuil:
    n=n+1
    c=0.8*c
  return n
Fns...|a|A|#|Outils|Exéc|Script
    
```

- Appuyer sur **F4** pour exécuter le script et observer l’affichage

```

PYTHON SHELL
>>>
>>> # Shell Reinitialized
>>> # L'exécution de WHILE
>>> from WHILE import *
>>> a(3.4,1)
6
>>> |
    
```

Conseil à l’enseignant : Il peut s’avérer nécessaire de devoir conserver les valeurs, lorsque celles-ci correspondent à un calcul qui doit être par exemple réutilisé ou simplement conservé afin d’être comparé (suite numérique par exemple). Dans ce cas l’utilisation des listes est pratique.

Appliquons nos connaissances : Les rebonds du ballon :

Une balle est lâchée d’une hauteur de 1,20 m et rebondit sur le sol des $\frac{3}{5}$ de la hauteur du rebond précédent.

Vous devez élaborer un algorithme donnant le nombre de rebonds au bout duquel la hauteur atteinte par la balle est strictement inférieure à 1 cm.

- Définir une variable H dans laquelle on stockera la hauteur du rebond exprimée en cm et R une variable pour compter les rebonds
- On devra répéter plusieurs fois l’instruction H prend la valeur $\frac{3}{5} \times H$ sans connaître à l’avance le nombre de répétitions.
- On testera donc si $H > 1$ et le traitement dans la boucle sera réalisé tant que cette condition restera vérifiée.



Algorithme

```

H ← 90
R ← 0
Tant que H ≥ 1
|   H ←  $\frac{3}{5} \times H$ 
|   R ← R + 1
Fin Tant que
    
```

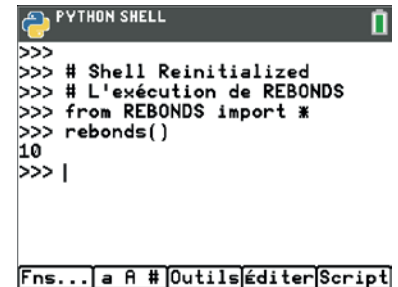
- Créer un nouveau script et le nommer rebonds
- On propose d’utiliser une fonction. Ainsi on veillera à respecter les indentations (une pour l’instruction **While** et une seconde pour l’instruction **Return** permettant de renvoyer le contenu de la variable R
- On peut également comme dans l’exemple n°1 créer une liste puis afficher les hauteurs successives des rebonds.

```

ÉDITEUR : REBONDS
LIGNE DE SCRIPT 0001
def rebonds():_
    H=120
    R=0
    while H>=1:
        H=H*3/5
        R=R+1
    return (R)
    
```

- Appuyer sur **F3** (Outils) et choisir le menu **2 : Indent** ← afin de fermer la boucle **While**, tout en conservant l'exécution de la fonction.
- Pour exécuter le script, appuyer sur **F4** (Exec)
- Entrer le nom de la fonction « rebonds() » et valider en appuyant sur la touche
- Enrichir éventuellement le script à l'aide d'un message, par exemple :

`return` (« Nombre de rebonds = »,R)



```
PYTHON SHELL
>>>
>>> # Shell Reinitialized
>>> # L'exécution de REBONDS
>>> from REBONDS import *
>>> rebonds()
10
>>> |
Fns... | a R # Outils | éditer | Script
```

Un défi : calculer la distance totale parcourue par la balle jusqu'à son immobilisation. (On supposera que les rebonds de la balle sont rigoureusement situés sur la verticale)

Unité 2 : Les boucles en programmation en Python

Application : Boucles et tests

Pour cette application de l'unité 2, on se propose de réinvestir les notions vues dans les leçons concernant les instructions conditionnelles ainsi que les boucles bornées et non bornées.

Objectifs :

- Utiliser la boucle **While** et **For** pour mettre en oeuvre un algorithme relatif à un problème de probabilités ou de statistiques.

Dans cette application, vous allez écrire un script permettant de :

- Obtenir un nombre aléatoire en créant une fonction **lancer**.
- Utiliser cette fonction dans un autre script afin de déterminer le nombre de lancers nécessaires pour obtenir une somme de 12 lors du lancer de 2 dés parfaitement équilibrés.
- Lors du lancer d'un seul dé, obtenir le nombre de fois où chaque face apparait afin éventuellement de calculer une fréquence à comparer à la probabilité de sortie de chaque face.



Lancer un dé.

- Le travail sur les nombres aléatoires nécessite le chargement du module **random** avant la définition de la fonction.
- Créer un nouveau script et le nommer LANCER
- Appuyer sur **F1** (Fns...) puis **[1]** (Modul) et choisir le module **2 : random**
- Définir la fonction **lancer()** permettant d'obtenir un nombre aléatoire entier entre 1 et 6.
- Tester votre script après avoir vérifié.
- Utiliser les touches de direction vers le haut (**[↑]**) afin de relancer le script.

```
ÉDITEUR : LANCER
LIGNE DE SCRIPT 0001
from random import *
def lancer():
    return randint(1,6)
```

Fns... | a A # | Outils | Exéc | Script

```
PYTHON SHELL
>>> lancer()
2
>>> lancer()
4
>>> lancer()
6
>>> lancer()
6
>>> lancer()
3
>>> |
```

Fns... | a A # | Outils | Éditer | Script

Nombre d'essais nécessaires.

On lance deux dés à 6 faces parfaitement équilibrés et on additionne les deux résultats obtenus. Vous allez écrire un autre script afin de modéliser les lancers de ces deux dés et rechercher le nombre *n* de lancers nécessaires afin d'obtenir la somme de 12.

De nombreuses solutions sont possibles, mais la première qui vient à l'esprit est de réutiliser la fonction précédente.

```
ÉDITEUR : DOUZE
LIGNE DE SCRIPT 0004
from random import *
def lancer():
    d=randint(1,6)
    return d
def somme():
    s=0
    n=0
    while s!=12:
        s=lancer()+lancer()
        n=n+1
    return n,s
```

Fns... | a A # | Outils | Exéc | Script

- La variable s donne la somme des lancers et n le nombre de lancers nécessaires avant d'atteindre 12.
- Toutes deux sont initialisées à 0
- En langage Python, le symbole \neq est obtenu en appuyant sur les touches `2nde` `math` ou bien à partir du menu **Fns...** puis **Ops** lors de l'édition du script. Ce symbole est représenté par `!=`
- Compléter le script en veillant à respecter l'indentation puis l'exécuter.
- Le premier nombre donne le nombre de lancers nécessaires pour atteindre la somme 12 affichée par le second.

```

PYTHON SHELL
>>> somme()
(14, 12)
>>> somme()
(36, 12)
>>> somme()
(15, 12)
>>> somme()
(7, 12)
>>> |
Fns... | a A # |Outils|Éditer|Script

```

Échantillonnage et fréquence

Vous venez d'observer sur l'exemple précédent que le nombre d'essais nécessaires avant d'obtenir un 12 fluctue. On peut donc être conduit à calculer la fréquence d'échantillonnage, qui pour un grand nombre d'essais doit tendre vers la probabilité théorique.

Vous allez dans ce dernier script :

Utiliser une boucle for pour répéter un seul lancer

- 1) Calculer le nombre de fois où une face apparaît.
- 2) Utiliser une liste, ce qui a l'avantage de rendre le script plus court.

- Créer un nouveau script et le nommer DE1
- Le résultat d'un lancer est stocké dans la liste `l` précédemment initialisée vide par l'instruction `l=[]`
- La liste `f` contient les numéros des faces.
- Ensuite un test est effectué sur la valeur de la variable (1 à 6) et chaque fois qu'une condition est vérifiée, une nouvelle liste `fl` est créée dans laquelle est stocké, pour chaque numéro de face variant de 1 à 6, le nombre d'occurrence observées.
- Modifier éventuellement le script afin de calculer la fréquence d'apparition de chaque face.

```

ÉDITEUR : DE1
LIGNE DU SCRIPT 0001
from random import *
l=[]
f=[1,2,3,4,5,6]
fl=[]
def lancer(n):
    for i in range(n):
        l.append(randint(1,6))
    for i in range(6):
        fl.append(l.count(f[i]))
    return fl
Fns... | a A # |Outils|Exéc|Script

```

Appuyer sur la touche **F4** afin d'exécuter le script. Observer les effectifs de sortie de chaque face, pour 100 lancers, les fréquences sont simples à calculer.

```

PYTHON SHELL
>>>
>>> # Shell Reinitialized
>>> # L'exécution de DE1
>>> from DE1 import *
>>> lancer(15)
[3, 6, 1, 2, 1, 2]
>>> |
Fns... | a A # |Outils|Éditer|Script

```


Unité 3 : Exemples d'applications

Compétence 1 : Fonctions et boucles

Dans cette première leçon de l'unité 3, vous allez mettre en oeuvre vos connaissances en algorithmique et en langage Python afin de :

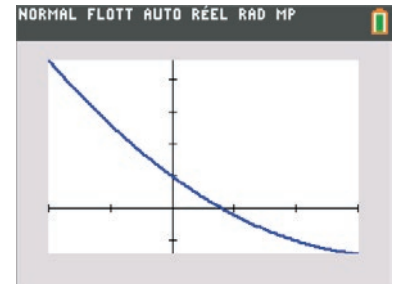
- Rechercher les solutions d'une équation $f(x) = 0$.
- Résoudre un problème d'optimisation.

Objectifs :

- Utiliser une fonction en langage Python.
- Mettre en oeuvre la boucle bornée **While**.

Principe de la dichotomie

On considère la fonction f définie sur l'intervalle $[-2,3]$ par $f(x) = x^2 - 7x + 5$
On utilise la calculatrice afin de tracer la courbe C_f représentant les variations de la fonction f .
Vous allez résoudre l'équation $f(x) = 0$ en écrivant un script Python correspondant à un algorithme connu et appelé « algorithme de dichotomie ».



Pour comprendre ce qu'est la dichotomie, on propose une petite expérience : « chercher un mot dans un gros dictionnaire papier de 1024 pages » :

- Vous l'ouvrez au milieu : le mot ne s'y trouve pas, mais il est avant (il est donc dans les 512 premières pages).
- Vous ouvrez la moitié de la 1ère moitié : le mot ne s'y trouve pas, mais il est après (il est donc dans les pages 257 à 512).
- Vous ouvrez la moitié de la 2ème moitié, etc...

A chaque fois que vous progressez, le nombre de pages qui reste à examiner est divisé par 2.

Ainsi, dans un dictionnaire de 1024 pages, vous êtes certain de trouver votre page en 10 recherches seulement, puisque $1024/(2^{10})=1$

Algorithme :

Tant que $b - a > prec$ faire :

$$m \leftarrow \frac{a+b}{2}$$

Si $f(m)$ et $f(a)$ sont de signe, opposés

$$b \leftarrow m$$

sinon

$$a \leftarrow m$$

Fin Tant que

Commentaires :

$[a, b]$: Bornes de l'intervalle d'étude

f : Fonction étudiée

On se place au milieu de l'intervalle $[a ; b]$

Si $f(m)$ et $f(a)$ sont de même signe, alors la solution de l'équation $f(x) = 0$ est située dans l'intervalle $[m; b]$

On se place donc sur $[m; b]$

sinon

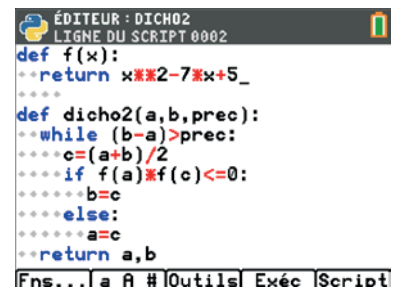
On se place sur $[a; m]$

Mise en œuvre de l'algorithme :

- Encadrer entre deux entiers la valeur x_0 solution de l'équation $f(x) = 0$ avec une précision donnée « $prec$ ».
- Vous remarquerez que $f(0) \times f(1) < 0$
- Calculer $f(0) \times f\left(\frac{1}{2}\right)$ et $f\left(\frac{1}{2}\right) \times f(1)$
- En déduire si x_0 appartient à l'intervalle $\left[0; \frac{1}{2}\right]$ ou $\left[\frac{1}{2}; 1\right]$

Créer un nouveau script et le nommer DICH0

- Entrer les différentes instructions, celles-ci se trouvent pour leur ensemble sous l'onglet (Fns...)
- Les tests peuvent être obtenus directement par l'appui sur les touches $\boxed{2^{nde}} \boxed{math}$
- Appuyer sur $\boxed{2^{nde}} \boxed{alpha}$ pour bloquer le clavier en mode alphanumérique.
- $[a, b]$ représente l'intervalle d'étude et n le nombre d'étapes.

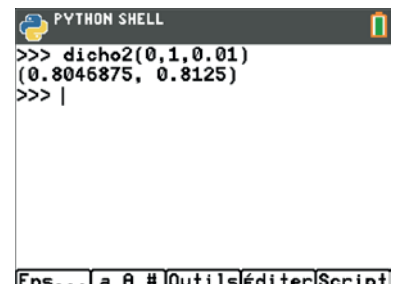


```

ÉDITEUR : DICH02
LIGNE DU SCRIPT 0002
def f(x):
    return x**2-7*x+5_
def dich02(a,b,prec):
    while (b-a)>prec:
        c=(a+b)/2
        if f(a)*f(c)<=0:
            b=c
        else:
            a=c
    return a,b
Fns... | a A # | Outils | Exéc | Script
    
```

Exécuter le script :

A partir de la représentation graphique de la fonction, tester le script.

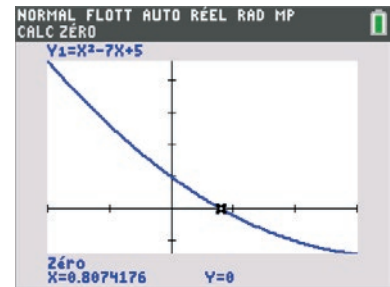


```

PYTHON SHELL
>>> dich02(0,1,0.01)
(0.8046875, 0.8125)
>>> |
Fns... | a A # | Outils | Éditer | Script
    
```

Affiner votre recherche afin de trouver une solution proche de celle qui est proposée par la calculatrice.

La valeur exacte de x_0 dans l'intervalle $[0 ; 1]$ étant donnée par : $x_0 = \frac{7-\sqrt{29}}{2}$



Prolongements possibles :

- a) Au lieu de donner la précision, on peut travailler avec le nombre d'étapes.

```
ÉDITEUR : DICH01
LIGNE DU SCRIPT 0010
def f(x):
  return x**2-7*x+5
...
def dich0(a,b,n):
  for i in range(n):
    c=(a+b)/2
    if f(a)*f(c)<=0:
      b=c
    else:
      a=c
  return a,b
Fns... a A # |Outils| Exéc |Script
```

```
PYTHON SHELL
>>> dich0(0,1,25)
(0.8074175715446472, 0.8074176013469696)
>>> |
Fns... a A # |Outils| Éditer |Script
```

La suite du script reste identique

- b) Envisager une programmation récursive

Remarque : Pour plus d'information sur la récursivité voir Compétence 3 de l'unité 3

```
def dich0(a,b,prec):
  if (b-a)<=prec:
    return a,b
  else:
    c=(a+b)/2
    if f(a)*f(b)<=0:
      return dich0(a,c,prec)
    else:
      return dich0(c,b,prec)
```

Unité 3 : Exemples d'applications

Compétence 2 : La boucle bornée FOR

Dans cette seconde leçon de l'unité 3, vous allez découvrir comment répéter un processus ou un ensemble d'instructions en utilisant une boucle bornée **FOR**.

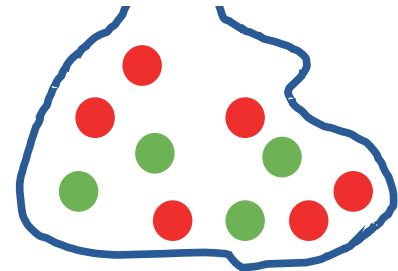
Objectifs :

- Appliquer une fonction.
- Découvrir et mettre en oeuvre la boucle bornée **FOR**.
- Utiliser la boucle **FOR** dans des exemples simples.

Constituer un échantillon :

Un sac opaque contient six jetons rouges et quatre jetons verts. On tire au hasard un jeton du sac, on note sa couleur puis on le replace dans le sac.

Programmer une fonction **couleur()** simulant une expérience aléatoire de variable.



- Quelles sont les valeurs possibles prises par la variable x ?
- On souhaite écrire un script qui permette de distinguer les boules rouges des vertes à l'aide d'un test.
- Commencer un nouveau script et le nommer « **ECHANTIL** »
- Comme vous travaillez sur des nombres aléatoires, le chargement de la bibliothèque « random » est nécessaire. Pour cela, appuyer sur l'onglet **F1 (Fns...)** et choisir le menu **Modul puis 2 : random...**
- Entrer le script ci-contre dans l'éditeur en veillant à respecter l'indentation.
- Afficher le résultat de la fonction dans la console.
- Exécuter le script plusieurs fois en appelant la fonction **couleur()**

```
ÉDITEUR : ECHANTIL
LIGNE DE SCRIPT 0001
from random import *
def couleur():
    x=randint(1,10)
    if x<=6:
        c="Rouge"
    else:
        c="Vert"
    return c
```

Fns... | a A # |Outils| Exéc |Script

```
PYTHON SHELL
>>> couleur()
'Rouge'
>>> couleur()
'Rouge'
>>> couleur()
'Rouge'
>>> couleur()
'Vert'
>>> couleur()
'Rouge'
>>> |
```

Fns... | a A # |Outils|Éditer|Script

Conseil à l'enseignant : ce script peut être modifié pour être appliqué sur un nombre quelconque de jetons. Dans ce cas-là, on pourra définir **couleur(n,a)** où n est le nombre de jetons et a le nombre de jetons rouges.

Appliquons nos connaissances : Échantillonnage et prise de décision :

On a réalisé un sondage à la sortie du nouveau spectacle proposé par un artiste. Ce sondage réalisé dans une grande ville montre que les deux tiers des personnes ayant vu le spectacle l'ont aimé. L'agent de l'artiste pense que toute la population française est dans le même état d'esprit. Il commande une enquête auprès d'un institut de sondage pour le vérifier.



Étude de la population :

Pour des besoins de l'enquête statistique, l'institut de sondage doit créer une fonction qui simule la réponse à la situation.

Votre travail consiste à créer cette fonction en respectant le cahier des charges suivant :

- Le spectacle a été apprécié, avec une probabilité $p = \frac{2}{3}$
- Le spectacle n'est pas apprécié, avec une probabilité $p = \frac{1}{3}$

1. Commencer un nouveau script et le nommer **PROBAS**
2. Écrire cette fonction dans l'éditeur et la tester plusieurs fois en appuyant sur **F4** (Exec) puis en tapant dans la console le nom de la fonction sans arguments : **question()**

```
ÉDITEUR : PROBAS
LIGNE DE SCRIPT 0001
from random import *
def question():
    s=randint(0,2)
    if s==0 or s==1:
        reponse=1
    else:
        reponse=0
    return reponse
Fns... a A # |Outils| Exéc |Script
```

Astuce : on peut utiliser la touche **VAR** puis flèche du haut pour répéter l'exécution, (c'est peut-être plus rapide que de taper le nom de la fonction...)

Simulation d'un échantillon de taille n :

L'institut de sondage souhaite simuler des échantillons de taille variable.

Vous devez donc créer dans le script courant une fonction **échantillon(n)** permettant de poser la question à un échantillon de taille n.

- Pour cela, créer une liste vide **L**
- Remplir cette liste en utilisant la fonction **question()** et en utilisant une boucle **For**

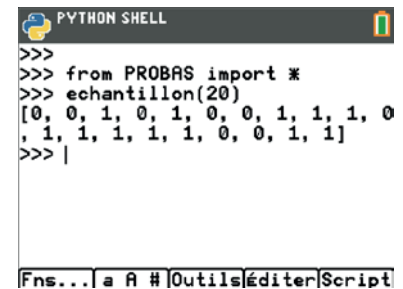
```
ÉDITEUR : PROBAS
LIGNE DU SCRIPT 0005
        reponse=1
    else:
        reponse=0
    return reponse

def echantillon(n):
    L=[]
    L=[question() for i in range
      (n)]
    return L
Fns... a A # |Outils| Exéc |Script
```

Conseil à l'enseignant : Le langage Python permet d'utiliser une fonction pour remplir une liste incrémentée par une boucle For, comme arguments de la liste

La réalisation de cet exemple est possible sans l'utilisation de listes. Les scripts sont alors à modifier légèrement en remplaçant les instructions relatives aux listes par des boucles bornées incrémentant une variable.

- Tester la fonction **echantillon(n)** pour un échantillon de taille 20, en exécutant le script et en tapant dans la console « echantillon(20) »



```

PYTHON SHELL
>>>
>>> from PROBAS import *
>>> echantillon(20)
[0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1]
>>> |
    
```

Algorithme

Validation de l'échantillon :

L'institut de sondage souhaite déterminer le pourcentage d'échantillons dont la fréquence de personnes qui ont apprécié le spectacle appartient à l'intervalle de fluctuation à 95% de $p = \frac{2}{3}$

A la suite du script précédent, il vous est demandé de créer deux fonctions :

- freq_echantillon(n)** qui calcule automatiquement la fréquence des réponses 1 observée dans l'échantillon L de taille n
- interv_fluctu(te,ne)** pour déterminer si la fréquence d'un échantillon de taille n est compris dans l'intervalle de fluctuation à 95%

Fonction **interv_fluctu**(taille ech , nbre ech)

```

Nf ← 0
Pour i allant de 1 à nbre ech faire
    f ← freq_echantillon(taille ech)
    Si f appartient à  $\left[ p - \frac{1}{\sqrt{n}} ; p + \frac{1}{\sqrt{n}} \right]$  alors
        Nf ← Nf + 1
    Fin si
Fin Pour
    
```

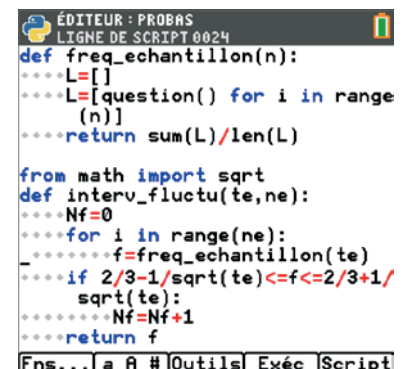
Conseil à l'enseignant : On rappelle que pour un caractère dont la proportion dans une population donnée est p.

Pour $n \geq 25$ et $0.2 \leq p \leq 0.8$, la fréquence du caractère dans les échantillons de taille n appartient à l'intervalle

$$\left[p - \frac{1}{\sqrt{n}} ; p + \frac{1}{\sqrt{n}} \right] \text{ dans 95\% des cas.}$$

Cet intervalle est appelé intervalle de fluctuation à 95%.

A partir de la fonction **interv_fluctu** écrit en langage naturel, vérifier que vous obtenez la fonction **interv_fluctu** en Python



```

ÉDITEUR : PROBAS
LIGNE DE SCRIPT 0024
def freq_echantillon(n):
    L=[]
    L=[question() for i in range(n)]
    return sum(L)/len(L)

from math import sqrt
def interv_fluctu(te,ne):
    Nf=0
    for i in range(ne):
        f=freq_echantillon(te)
        if 2/3-1/sqrt(te)<=f<=2/3+1/sqrt(te):
            Nf=Nf+1
    return f
    
```



- Tester le script plusieurs fois pour un échantillon de taille 100
- En déduire pour un exemple, l'intervalle de fluctuation à 95%
- Cette étude statistique remet-elle en question l'affirmation de l'agent de l'artiste ?

```
PYTHON SHELL
Attendez s'il vous plaît...
PROBAS...
Configuration terminée

>>> from PROBAS import *
>>> interv_fluctu(100,100)
0.75
>>> interv_fluctu(100,100)
0.65
>>> |
Fns... | a A # | Outils | Éditer | Script
```



Unité 3 : Exemples d'applications

Compétence 3 : Programmation et récursivité

Dans cette troisième leçon de l'unité 3, vous allez utiliser les fonctions afin de réaliser une programmation récursive.

Objectifs :

- Appliquer une fonction.
- Découvrir et mettre en oeuvre la programmation récursive.

Calcul de PGCD (Méthode itérative)

Pour calculer le « plus grand commun diviseur de deux nombres » (PGCD), on utilise l'algorithme d'Euclide.

Remarque : $a > b$

On procède de la manière suivante :

- On effectue la division euclidienne de a par b . On note r le reste (on n'utilise pas le quotient).
- On remplace ensuite a par b et b par r .
- **Tant que** le reste est différent de 0, on **réitère** le procédé.

Après un certain nombre d'itérations, on obtient un reste égal à 0.

Le PGCD de a et de b est alors le reste précédent (c'est à dire **le dernier reste non nul**).

- Créer un nouveau script et le nommer pgcd
- Créer une fonction pgcd(a,b) en tapant sur F1 menu (Fonc)
- Le symbole \neq s'écrit en langage Python à l'aide de $!=$ et se trouve dans le menu F2 (a A...) ou bien dans le menu tests de la calculatrice.
- Noter l'affectation $a,b=b,r$ dans le script qui permet de gagner une ligne de code, mais qui correspond à la réalisation des affectations $a = b$ et $b = r$

- Exécuter le script pour différents nombres

```
ÉDITEUR : PGCD
LIGNE DU SCRIPT 0002
#Calcul itératif du pgcd de a e
t b
def pgcd(a,b):
..while b!=0:
...r=a%b
...a,b=b,r
..return a

Fns... | a A # |Outils| Exéc |Script
```

```
PYTHON SHELL
PYTHON01...
Configuration terminée.

>>>
>>> # Shell Reinitialized
>>> # L'exécution de PGCD
>>> from PGCD import *
>>> pgcd(56,42)
14
>>> |

Fns... | a A # |Outils|Éditer|Script
```


Un pas plus loin .

3 : Programmation récursive :

Un algorithme est dit récursif si, à un moment, il s'appelle lui-même.

La récursivité peut posséder de nombreux avantages dans un algorithme. Premièrement, elle permet de résoudre des problèmes, d'habitude insolubles avec l'utilisation de simples boucles *pour* ou *tant que*. Elle peut aussi rendre un algorithme plus lisible et plus court, mais surtout, elle permet, dans certains cas, un gain colossal de temps comme c'est le cas dans les algorithmes de tri.

Un premier exemple de récursivité :

- Créer un nouveau script et le nommer **REC1**
 - Entrer le script ci-contre
 - Quel est le cas de base dans cette fonction récursive ?
 - Qu'est ce qui garantit dans cette fonction récursive, que le script finira par s'arrêter ?
 - Écrire le processus complet
-
- Exécuter le script dans une console
 - Que retourne $f(a,b)$ a et b étant des entiers naturels non nuls ?

```
ÉDITEUR : REC1
LIGNE DU SCRIPT 0004
def f(a,b):
    if b==0:
        return a
    return a+f(a,b-1)

```

```
PYTHON SHELL
>>>
>>> # Shell Reinitialized
>>> # L'exécution de REC1
>>> from REC1 import *
>>> f(3,5)
18
>>> |

```

3.1 : Un calcul de pgcd récursif :

Le calcul du PGCD de deux entiers positifs a et b utilise toujours l'algorithme d'Euclide. Soit r le reste de la division euclidienne de a par b :
 $a = b*q + r, r < b$.

Tout diviseur commun de a et b divise aussi $r = a - b*q$ et réciproquement tout diviseur commun de b et r divise aussi $a = b*q + r$. Donc le calcul du PGCD de a et b se ramène à celui du PGCD de b et r ; et on peut recommencer sans craindre une boucle sans fin, car les restes successifs sont strictement décroissants. Le dernier reste non nul obtenu est le PGCD cherché.

```
ÉDITEUR : PGCD2
LIGNE DU SCRIPT 0006
def pgcd_recuratif(a,b):
    r=a%b
    if r==0:
        return b
    else:
        return pgcd_recuratif(b,r)

```

Par exemple si $a=96$ et $b=81$, les calculs sont les suivants :

et le PGCD est 3

a		b		r
96	=	1 * 81	+	15
81	=	5 * 15	+	6
15	=	2 * 6	+	3
6	=	2 * 3	+	0

- Exécuter le script en appuyant sur la touche **F4** (Exec) et le tester pour quelques valeurs.

```
PYTHON SHELL
>>> pgcd_recuratif(910,105)
35
>>> |
Fns... | a A # | Outils | Éditer | Script
```

Conseil à l'enseignant : Pour éviter les cercles vicieux, une fonction récursive doit toujours comporter un cas particulier où le résultat est calculé directement, c'est à dire sans appel récursif ; il faut aussi s'assurer que ce cas particulier finira toujours par se présenter.

Unité 3 : Exemples d'applications

Application : Tests, boucles

Dans cette application de l'unité 3, vous allez utiliser les notions acquises dans les leçons précédentes afin de programmer des algorithmes vous permettant d'affiner vos connaissances des nombres et en particulier des nombres premiers.

Objectifs :

- Mettre en oeuvre les boucles et tests pour la programmation complète d'un algorithme en Python

Un nombre entier est dit premier s'il possède exactement deux diviseurs : 1 et lui-même.

Par exemple :

- 1 n'est pas premier (il ne possède qu'un seul diviseur : 1)
- 7 est un nombre premier (ses diviseurs sont 1 et 7).
- 8 n'est pas premier (il possède quatre diviseurs : 1, 2, 4 et 8)

Les nombres premiers sont : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, ...
Il en existe une infinité.

On se propose de déterminer le 2019^{ème} nombre premier.

On considère l'algorithme ci-contre.

- Afin de comprendre l'algorithme, à quelle condition sur les nombres 2, 3, $n-1$, un entier $n \geq 2$ est-il premier ?
- Réaliser l'écriture de la fonction « ep » qui à tout entier naturel n devra renvoyer 1 si n est premier et 0 sinon.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

```

Si n ≤ 1 alors
    | Retourner 0
Fin si
Pour k de 2 à n-1 Faire
    | Si n%k = 0 Alors
        | | Retourner 0
    | Fin si
Fin pour
Retourner 1
    
```

La partie principale du programme est donnée par l'algorithme ci-contre.

Votre travail consiste à implémenter cet algorithme en langage Python afin de répondre au problème posé

```

N ← 2
no ← 1
Tant que no < 2019 Faire
    | N ← N + 1
    | No ← no + ep(N)
Fin Tant que
Afficher « Le 2019e nombre premier est », N
    
```

- Commencer un nouveau script et le nommer **NBPREM**
- Entrer les différentes instructions en veillant à respecter l'indentation

```
ÉDITEUR : PYTHON01
LIGNE DU SCRIPT 0001
from math import *
def ep(n):
    if n<=1:
        return 0
    for k in range (2,floor(sqrt(n)
    )+1):
        if n%k==0:
            return 0
    return 1
N=2
no=1
while no<2019:
    N=N+1
    no=no+ep(N)
print(N)_

Fns... | a A # | Outils | Exéc | Script
```

- Appuyer sur la touche F4 pour exécuter le script (temps de calcul très long sur la calculatrice environ 10 minutes)
- Vérifier que le 2019^e nombre premier est bien **17569**

```
PYTHON SHELL
>>>
>>> # L'exécution de PYTHON01
>>> from PYTHON01 import *
17569
>>> |

Fns... | a A # | Outils | Éditer | Script
```

Conseil à l'enseignant : en changeant le test de primalité utilisé et en ne testant que sur 2 à la partie entière de racine (n)+1 (ce qui assure de bien tester tous les diviseurs potentiels) , on diminue considérablement le temps d'attente qui passe à quelques secondes.



[education.ti.france](https://www.facebook.com/education.ti.france)



[@TIEducationFR](https://twitter.com/TIEducationFR)



[TledtechFR](https://www.youtube.com/TledtechFR)



Contactez notre service client TI-Cares
education.ti.com/fr/csc
01 41 04 60 40

education.ti.com/fr