

Introduction à la Programmation sous Python

Plan

- *Introduction*
- *Données et variables*
- *Contrôle du flux d'exécution*
- *Instructions répétitives*
- *Principaux types de données*
- *Fonctions prédéfinies*
- *Fonctions originales*
- *Manipuler des fichiers*
- *Approfondir les structures de données*
- *Classes, objets, attributs*
- *Classes, méthodes, héritage*

Introduction

Most Pull Requests 2017

GitHub

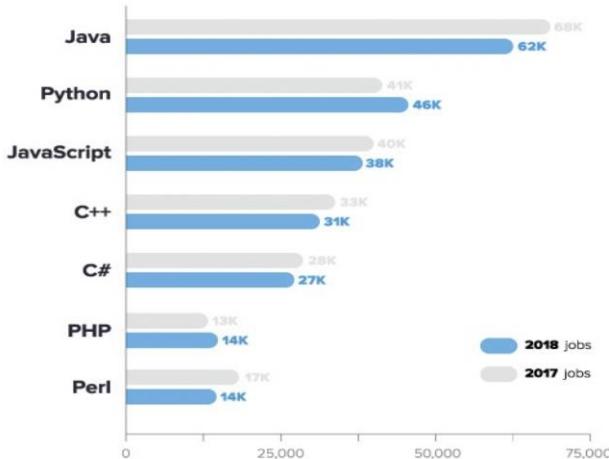
Javascript	2,300,000
Python	1,000,000
Java	986,000
Ruby	870,000
PHP	559,000
C++	413,000
CSS	335,000
C#	326,000
Go	285,000
C	239,000
TypeScript	207,000
Shell	206,000
Swift	107,000
Scala	99,000
Objective-C	66,000

4

Introduction

Job postings containing top languages

Indeed.com - November, 17th 2017



5

Introduction

- Langage dynamiquement interprété
- Développé et maintenu depuis 1989 par Guido Van Rossum
- Open source pour interpréteur et bibliothèque standard
- Orienté objet
- Facile d'utilisation (d'apprehension)
- Système de gestion d'exceptions
- Nombreux types d'utilisation/bibliothèque spécialisées

6

Calculer avec Python en mode interactif

L'interpréteur peut être lancé directement depuis la ligne de commande (dans un « shell » Linux, ou bien dans une fenêtre DOS sous Windows) : taper la commande 'python' ou 'python2' ou 'python3'

```

Terminal
Eichier Édition Affichage Terminal Aide
fred@newton:~$ python3
Python 3.1.1+ (r311:74480, Nov  2 2009, 14:49:22)
[GCC 4.4.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>

Python Shell
File Edit Shell Debug Options Windows Help
Python 3.1.1 (r311:74483, Aug 17 2009, 17:02:12) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> |

```

Les trois caractères >>> constituent le **signal d'invite**, ou **prompt principal**, lequel vous indique que Python est prêt à exécuter une commande.

7

Calculer avec Python en mode interactif

1. utiliser l'interpréteur comme une simple calculatrice de bureau.

2. tester des commandes, comme :

>>> 5+3

>>> 2 - 9

les espaces sont optionnels

>>> 7 + 3 * 4

la hiérarchie des opérations mathématiques

est-elle respectée ?

>>> (7+3)*4

Les parenthèses sont fonctionnelles.

>>> 20 / 3

une division entière

>>> 20 % 3

>>> 20.5 / 3

>>> 20,5 / 3

Erreur

8

Structuration et notion de bloc

- En Python, chaque instruction s'écrit sur une ligne sans mettre d'espace au début
- Ces instructions simples peuvent cependant être mises sur la même ligne en les séparant par des points virgules (;), les lignes étant exécutées dans l'ordre de gauche à droite
- La séparation entre les en-têtes qui sont des lignes de définition de boucles, de fonction, de classe se terminent par les deux points (:)
- Le contenu ou « bloc » d'instructions correspondant se fait par indentation des lignes
- Une indentation s'obtient par le bouton tab (pour tabulation) ou bien par 4 espaces successifs
- L'ensemble des lignes indentées constitue un bloc d'instructions

```
a = 10
b = 3
print(a, b)
```

```
a = 10; b = 3; print(a, b)
```

Noms de variables et mots réservés

Les noms de variables sont des noms qu' on choisit assez librement

De préférence assez courts, mais aussi explicites que possible, pour exprimer clairement ce que la variable est censée contenir :

*altitude, altit ou alt (au lieu de x) pour exprimer une altitude
prix_unit pour exprimer un prix unitaire, etc.*

9

Noms de variables et mots réservés

Quelques règles pour les noms de variables sous Python :

1. Un nom de variable est une séquence de lettres (a à z , A à Z) et de chiffres (0 à 9), qui doit toujours commencer par une lettre.
2. Seules les lettres ordinaires sont autorisées. Les lettres accentuées, les cédilles, les espaces, les caractères spéciaux tels que \$, #, @, etc. sont interdits, à l'**exception** du caractère _ (souligné).
3. La **casse est significative** (les caractères majuscules et minuscules sont distingués). **Attention** : Joseph, joseph, JOSEPH sont donc des variables différentes. Soyez attentifs !
4. Prenez l'habitude d'écrire l'essentiel des noms de variables en caractères minuscules (y compris la première lettre). Il s'agit d'une simple convention, mais elle est largement respectée. N'utilisez les majuscules qu'à l'intérieur même du nom, pour en augmenter éventuellement la lisibilité, comme dans *TableDesMatieres*.

10

Noms de variables et mots réservés

En plus de ces règles, il faut encore ajouter que vous ne pouvez pas utiliser comme noms de variables les 29 « mots réservés » au langage ci-dessous :

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

11

Affectation (ou assignation)

Nous savons désormais comment choisir judicieusement un nom de variable. Voyons à présent comment nous pouvons en définir et *affecter* une valeur.

En Python comme dans de nombreux autres langages, l'opération d'affectation est représentée par le signe *égal* :

```
>>> n = 7                      # donner à n la valeur 7
>>> msg = "Quoi de neuf ?"      # affecter la valeur "Quoi de neuf ?" à msg
>>> pi = 3.14159                # assigner sa valeur à la variable pi
```



12

Afficher la valeur d'une variable

Pour afficher la valeur à l'écran, il existe deux possibilités. :

- La première consiste à entrer au clavier le nom de la variable, puis <Enter>.

```
>>> n
7
>>> msg
"Quoi de neuf ?"      #affiche les guillemets, donc le type.
>>> pi
3.14159
```

- A l'intérieur d'un programme, vous utiliserez toujours l'instruction **print** :

```
>>> print (msg)
Quoi de neuf ?      #pas de guillemets.
>>> print(n)
7
```

13

Afficher la valeur d'une variable

Pas de Typage explicite des variables sous python

- Sous Python, il n'est pas nécessaire d'écrire des lignes de programme spécifiques pour définir le type des variables avant de pouvoir les utiliser.
- Il vous suffit en effet d'assigner une valeur à un nom de variable pour que celle-ci soit automatiquement créée avec le type qui correspond au mieux à la valeur fournie.
- Par exemple, les variables n, msg et pi ont été créées automatiquement chacune avec un type différent:
 - « nombre entier » pour n,
 - « chaîne de caractères » pour msg,
 - « nombre à virgule flottante » ou « float », pour pi

14

Affectations multiples

Sous Python, on peut assigner une valeur à plusieurs variables simultanément.

Exemple :

```
>>> x = y = 7
>>> x
7
>>> y
7
```

On peut aussi effectuer des **affectations parallèles** à l'aide d'un seul opérateur :

```
>>> a, b = 4, 8.33
>>> a
4
>>> b
8.33
```

Dans cet exemple, les variables **a** et **b** prennent simultanément les nouvelles valeurs 4 et 8,33.

15

Types de données

- Les types de données les plus utilisés sont :

Type entier (integer)

```
a=1
b=555
c=6
```

Type réel (float)

```
b=0.003
b=2.
```

Type Boolean

```
b=True
C=False
```

Type caractère

```
phrase1="ecole"
Phrase2="ems\u00e9"
Phrase3="j'aime bien"
```

Exercices

1.Décrivez le plus clairement et le plus complètement possible ce qui se passe à chacune des trois lignes de l'exemple ci-dessous :

```
>>> largeur = 20
>>> hauteur = 5 * 9.3
>>> largeur * hauteur
930
```

2.Assignez les valeurs respectives 3, 5, 7 à trois variables a, b, c.

Effectuez l'opération a - b/c.

16

Opérateurs et expressions

On manipule les valeurs et les variables qui les réferencent, en les combinant avec des **opérateurs** pour former des **expressions**.

Exemple :

a, b = 7.3, 12
y = 3*a + b/5

Exercice

Décrivez ce qui se passe à l' exécution des lignes suivantes :

```
>>> r , pi = 12, 3.14159
>>> s = pi * r**2
>>> s
>>> type(r), type(pi), type(s)
>>>
```

Quelle est, à votre avis, l'utilité de la **fonction type()** ?

Opérateur	nom
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo
**	Puissance
//	Division entière

17

Opérateurs d'affectation composés

- Python reconnaît également des opérateurs d'affectation qu'on appelle "composés" et qui vont permettre d'effectuer deux opérations à la suite : une première opération de calcul suivie immédiatement d'une opération d'affectation:

Opérateur	Utilisation	Explication
<code>+=</code>	<code>X+=1</code>	Ajoute 1 à la dernière valeur connue de x et affecte la nouvelle valeur(ancienne + 1)à x
<code>-=</code>	<code>X-=1</code>	Enlève 1 à la dernière valeur connue de x et affecte la nouvelle valeur à x
<code>*=</code>	<code>X*=2</code>	Multiplie par 2 la dernière valeur connue de x et affecte la nouvelle valeur à x
<code>/=</code>	<code>x/=2</code>	Divise par 2 la dernière valeur connue de x et affecte la nouvelle valeur à x
<code>%=</code>	<code>X%=2</code>	Calcule le reste de la division entière de x par 2 et affecte ce reste à x
<code>//=</code>	<code>x//=2</code>	Calcule le résultat entier de la division de x par 2 et affecte ce résultat à x
<code>**=</code>	<code>X**=4</code>	Elève x à la puissance 4 et affecte la nouvelle valeur dans x

Priorité des opérations

PEMDAS pour le mémoriser

- P** pour **parenthèses**. Ce sont elles qui ont la plus haute priorité. Elles vous permettent donc de « forcer » l'évaluation d'une expression dans l'ordre que vous voulez. Ainsi $2 * (3 - 1) =$, et $(1 + 1) ** (5 - 2) =$.
- E** pour **exposants**. Les exposants sont évalués avant les autres opérations. Ainsi $2 ** 1 + 1 =$ et $3 * 1 ** 10 =$
- M** et **D** pour **multiplication** et **division**, qui ont la même priorité. Elles sont évaluées avant **l'addition A** et la **soustraction S**, lesquelles sont donc effectuées en dernier lieu. Ainsi $2 *$, et $2 / 2 - 1 =$
- Si deux opérateurs ont la même priorité, l'évaluation est effectuée de gauche à droite. Ainsi dans l'expression $30 * 100 / 60$, la multiplication est effectuée en premier, et la machine doit donc ensuite effectuer $3000 / 60$, ce qui donne **50**.

Composition

L'une des grandes forces d'un langage de programmation de haut niveau est qu'il permet de **construire des instructions complexes par assemblage de fragments divers**. Ainsi par exemple, si vous savez comment additionner deux nombres et comment afficher une valeur, vous pouvez combiner ces deux instructions en une seule :

```
>>> print(17 + 3)
```

20

Cela n'a l'air de rien, mais cette fonctionnalité qui paraît si évidente va vous permettre de programmer des algorithmes complexes de façon claire et concise.
Exemple :

```
>>> h, m, s = 15, 27, 34
```

```
>>> print("nombre de secondes écoulées depuis minuit = ", h*3600 + m*60 + s)
```

nombre de secondes écoulées depuis minuit = 55654

Ce que vous placez à la gauche du signe *égale* dans une expression doit toujours être une variable, et non une expression : le signe égale n'a pas la même signification qu'en mathématique, il s'agit d'un symbole d'affectation

m + 1 = b est incorrect.

Par contre, a = a + 1 est inacceptable en math,
mais correct en programmation

19

Opération d'entrée

- La fonction `input()` retourne une valeur qui correspond à ce que l'utilisateur a entré. Cette valeur peut alors être assignée à une variable quelconque
- `Input` est une fonction qui renvoie toujours une chaîne de caractères
- Pour changer le type d'une variable, on utilise :
 - `int()`: pour les entiers
 - `float()`: pour les nombres à virgule flottante

```
prénom=input("entrez votre nom: ")
print ("bonjour: ", prénom)
```

Opération d'entrée

```
print ("veuillez saisir un nombre positif")
nn= input()
print("la carré de ", int(nn), "vaut", int(nn)*int(nn))
```

Conversion de la sortie en entier

Affichage:
veuillez saisir un nombre positif
5
la carré de 5 vaut 25

```
print ("veuillez saisir un nombre positif")
nn= input()
print("la carré de ", float(nn), "vaut", float(nn)*float(nn))
```

Conversion de la sortie en float
Affichage:
veuillez saisir un nombre positif
5
la carré de 5.0 vaut 25.0

Opération de sortie

- La fonction **print()** de Python est souvent utilisée pour afficher des variables et des chaînes de caractères
- Pour combiner à la fois du texte et une variable, Python utilise le caractère +:
- Pour tronquer une chaîne de caractères à afficher il est possible d'utiliser \n

```
x="Python est "
y="cool"
z=x+y
print(z)
```

Ce bloc affiche:
Python est cool

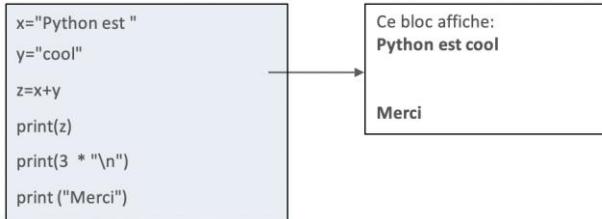
```
print ("Bonjour \nPython")
```

Ce bloc affiche:
Bonjour
Python

Opération de sortie

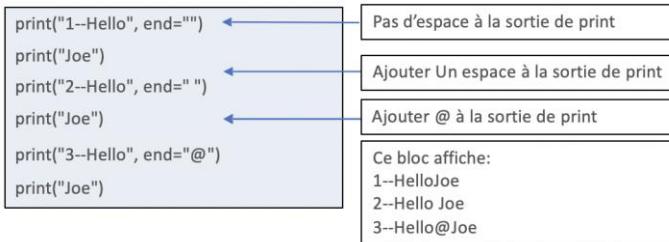
Pour imprimer des lignes vides, il est possible d'utiliser l'une des méthodes suivantes :

- Utiliser un chiffre représentant le nombre de lignes vide suivi de « * » et \n:
- Remplacer le nombre des lignes vierges par des \n



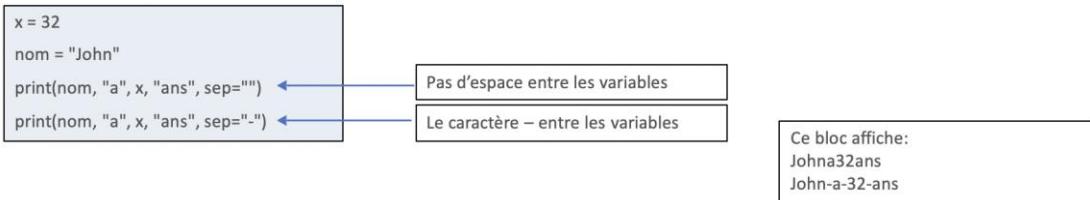
Opération de sortie

- La fonction « **end** » permet d'ajouter n'importe quelle chaîne à la fin de la sortie de la fonction **print**



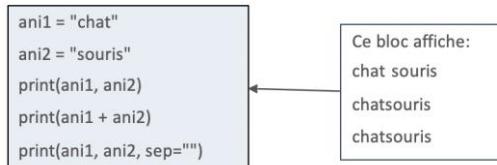
Opération de sortie

- Le mot clé « **sep** » précise une séparation entre les variables chaînes



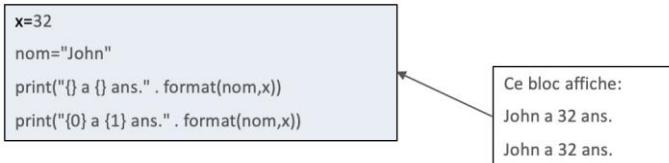
Opération de sortie

- Pour afficher deux chaînes de caractères l'une à côté de l'autre, sans espace, on peut soit les concaténer, soit utiliser l'argument par mot-clé **sep** avec une chaîne de caractères vide :



Opération de sortie

- La méthode **.format()** permet une meilleure organisation de l'affichage des variables dans une chaîne de caractères :



Exercices

Écrivez les programmes suivants en Python:

- demandez à un utilisateur son prénom et son âge et les affiche à l'écran
- calculer la surface d'un rectangle
- Calculer le périmètre d'un cercle
- Calculer le salaire net d'un salarié :
 - Données en entrée :
 - Nom et prénom d'un salarié
 - Âge du/de la salarié
 - Salaire et prime du/de la salarié
 - Résultat attendu en sortie : affichage à l'écran des messages suivants:
 - Prénom Nom a Âge (augmenté de 1) ans.
 - Prénom Nom gagne Salaire total (salaire + prime * 1,9).
- Convertir un nombre de secondes en h:m:s

Sélection ou exécution conditionnelle

• C'est une technique permettant d'aiguiller le déroulement du programme dans différentes directions, en fonction des circonstances rencontrées.

• Disposer d'instructions capables de tester une certaine condition et de modifier le comportement du programme en conséquence.

L'instruction if.

```
>>> a = 150
>>> if (a > 100):
...     print("a dépasse la centaine")
...
```

Tabulation obligatoire

En interactif :

Frappez encore une fois <Enter>. Le programme s'exécute, et vous obtenez :
a dépasse la centaine.

Recommencez le même exercice, mais avec a = 20 en guise de première ligne : cette fois Python n'affiche plus rien du tout.

Sélection ou exécution conditionnelle

```
>>> a = 20
>>> if (a > 100):
...     print("a dépasse la centaine")
... else:
...     print("a ne dépasse pas cent")
...
```

En interactif :

Frappez <Enter> encore une fois. Le programme s'exécute, et affiche cette fois :
a ne dépasse pas cent.

Comme vous l'aurez certainement déjà compris, l'instruction **else** (« sinon », en anglais) permet de programmer une exécution alternative, dans laquelle le programme doit choisir entre deux possibilités

Sélection ou exécution conditionnelle

On peut faire mieux encore en utilisant aussi l'instruction **elif** (contraction de « else if ») :

```
>>> a = 0
>>> if a > 0 :
...     print("a est positif")
... elif a < 0 :
...     print("a est négatif")
... else:
...     print("a est nul")
```

Opérateurs de comparaison

La condition évaluée après l'instruction «if» peut contenir les **opérateurs de comparaison** suivants :

<code>x == y</code>	# x est égal à y (deux signes « égale » et non d'un seul)
<code>x != y</code>	# x est différent de y
<code>x > y</code>	# x est plus grand que y
<code>x < y</code>	# x est plus petit que y
<code>x >= y</code>	# x est plus grand que, ou égal à y
<code>x <= y</code>	# x est plus petit que, ou égal à y

Même symbolisme qu' en C++ et en Java

Exemple :

```
>>> a = 7
>>> if (a % 2 == 0):
...     print("a est pair")
...     print("parce que le reste de sa division par 2 est nul")
... else:
...     print("a est impair")
... 
```

Instructions composées – Blocs d'instructions

L'instruction `if` est votre premier exemple d'*instruction composée*

Sous Python, toutes les instructions composées ont toujours la même structure : une ligne d'en-tête terminée par un double point, suivie d'une ou de plusieurs instructions indentées sous cette ligne d'en-tête.

Exemple :

```
Ligne d'en-tête:  
    première instruction du bloc  
    ...  
    ...  
    dernière instruction du bloc
```

S'il y a plusieurs instructions indentées sous la ligne d'en-tête, **elles doivent l'être exactement au même niveau** (comptez un décalage de 4 caractères, par exemple). Ces instructions indentées constituent ce que nous appellerons désormais un **bloc d'instructions**.

Instructions imbriquées

Il est parfaitement possible d'imbriquer les unes dans les autres plusieurs instructions composées, de manière à réaliser des structures de décision complexes.

Exemple :

```
if embranchement == "vertébrés": #1
    if classe == "mammifères": #2
        if ordre == "carnivores": #3
            if famille == "félins": #4
                print("c'est peut-être un chat") #5
                print("c'est en tous cas un mammifère") #6
            elif classe == "oiseaux": #7
                print("c'est peut-être un canari") #8
        print("la classification des animaux est complexe" ) #9
```

Ce programme n'imprime la phrase « c'est peut-être un chat » que dans le cas où les quatres premières conditions testées sont vraies.

Pour que la phrase « c'est peut-être un canari » soit affichée, il faut que la variable `embranchement` contienne « vertébrés », et que la variable `classe` contienne « oiseaux ».

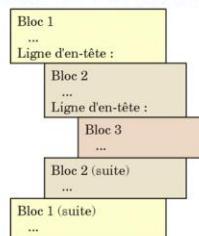
Quelques règles de syntaxe Python

- **Les limites des instructions et des blocs sont définies par la mise en page**

Avec Python, vous devez utiliser les sauts à la ligne et l'indentation. Python vous force donc à écrire du code lisible, et à prendre de bonnes habitudes que vous conserverez lorsque vous utiliserez d'autres langages.

- **Instruction composée = En-tête , double point , bloc d'instructions indenté**

Les blocs d'instructions sont toujours associés à une ligne d'en-tête contenant une instruction bien spécifique (if, elif, else, while, def, ...) se terminant par un double point



- **Les espaces et les commentaires sont normalement ignorés**

Instructions répétitives.

Ré-affectation

Permet de remplacer l'ancienne valeur d'une variable par une nouvelle

```
>>> altitude = 320
>>> altitude
320
>>> altitude = 375
>>> altitude
375
```

Rappels sur l'affectation :

- L'égalité est *commutative*, alors que l'affectation ne l'est pas.
- Les écritures `a = 7` et `7 = a` sont équivalentes, alors qu'une instruction de programmation telle que `375 = altitude` serait illégale.
- l'égalité est permanente, alors que l'affectation ne l'est pas.

```
>>> a = 5
>>> b = a # a et b contiennent des valeurs égales
>>> b = 2 # a et b sont maintenant différentes
```

Instructions répétitives

Exercice

Rappelons ici que Python permet d'affecter leurs valeurs à plusieurs variables simultanément :

```
>>> a, b, c, d = 3, 4, 5, 7
```

Supposons par exemple que nous voulions maintenant échanger les valeurs des variables **a** et **c**. (Actuellement, **a** contient la valeur 3, et **c** la valeur 5. Nous voudrions que ce soit l'inverse). Comment faire ?

Écrivez les lignes d'instructions nécessaires pour obtenir ce résultat sous python.

Répétitions en boucle - l'instruction while

L'une des tâches que les machines font le mieux est la répétition sans erreur de tâches identiques !

Une des méthodes pour programmer ces tâches répétitives est construite autour de l'instruction **while**.

1. Entrer les commandes ci-dessous

```
>>> a = 0
>>> while (a < 7): # (n'oubliez pas le double point !)
...     a = a + 1 # (n'oubliez pas l'indentation !)
... a
```

2. Frappez sur <Enter>

3. Que se passe-t-il ?

Répétitions en boucle - l'instruction while

```
>>> a = 0
>>> while (a < 7): # (n'oubliez pas le double point !)
...     a = a + 1 # (n'oubliez pas l'indentation !)
...a
```

Commentaires

1. Dans notre exemple, si la condition `a < 7` est encore vraie, le corps de la boucle est exécuté une nouvelle fois et le bouclage se poursuit.
2. La variable évaluée dans la condition doit exister au préalable (Il faut qu'on lui ait déjà affecté au moins une valeur)
3. Si la condition est fausse au départ, le corps de la boucle n'est jamais exécuté
4. Si la condition reste toujours vraie, alors le corps de la boucle est répété indéfiniment.

Exemple de boucle sans fin (à éviter) :

```
>>> n = 3
>>> while n < 5:
...     "hello!"
```

31

Exercices – Les conditions

1. Écrivez un programme pour vérifier si un nombre est pair ou impair
2. Écrivez un programme pour vérifier si un nombre est divisible par 3 et 13 ou non.
3. Écrivez un programme pour vérifier si l'année donnée par l'utilisateur est bissextile ou non. Année bissextile c'est une année spéciale contenant un jour supplémentaire, soit un total de 366 jours dans une année. Une année est considérée comme une année bissextile si l'année est exactement divisible par 4 mais non divisible par 100. L'année est également une année bissextile si elle est exactement divisible par 400.
4. Écrivez un programme pour entrer le numéro du mois entre (1-12) et afficher le nombre de jours de ce mois.
5. Écrivez un programme pour saisir un caractère de l'utilisateur et vérifiez si le caractère donné est un alphabet, un chiffre ou un caractère spécial
6. Écrivez un programme pour vérifier si un alphabet est une voyelle ou une consonne. Les lettres a, e, i, o et u en minuscules et en majuscules sont appelées voyelles. Les alphabets autres que les voyelles sont appelés consonnes.

Exercices – Les boucles

1. Écrivez un programme pour trouver la somme de tous les entiers entre 1 et n
2. Écrivez un programme qui demande un nombre et calcule sa factorielle.
3. Écrire un programme qui affiche la table de multiplication d'un nombre donné
4. Écrivez un programme qui vérifie si un nombre donné est premier ou non.
5. Écrivez un programme qui demande deux nombres de l'utilisateur et trouve le plus grand diviseur commun.
6. Écrivez un programme qui compte le nombre de chiffres d'un entier donné en utilisant une boucle.
7. Écrivez un programme qui demande un nombre et calcule la somme de ses chiffres en utilisant une boucle.
8. Ecrivez un programme qui affiche la suite de symboles suivante :

```

*
**
***
****
*****
******

```

Structures itératives : La boucle « for »

- Principes de la boucle for:
 - Elle est utilisée pour itérer sur une séquence
 - Elle ne s'applique que sur une collection de valeurs. Ex. tuples, listes,... à voir plus tard.
- La fonction range()
 - Pour parcourir un ensemble d'instructions un nombre spécifié de fois, nous pouvons utiliser la fonction **range ()**.
 - La fonction range () est une suite arithmétique simple qui renvoie une séquence de nombres, commençant à **0 par défaut**, et incrémentant de **1 (par défaut)**, et se termine à un nombre spécifié

(1) `range(4)` → 0 1 2 3
 (2) `range(1, 4)` → 1 2 3
 (3) `range(0, 5, 2)` → 0 2 4

La boucle « for »

Séquence est une collection de valeurs
Peut être générée avec range()

```
for indice in séquence:  
    bloc d'instructions
```

- Remarques :

- Attention à l'**indentation** toujours
- On peut « casser » la boucle avec **break**
- On peut passer directement à l'itération suivante avec **continue**
- Des boucles imbriquées sont possibles
- Le bloc d'instructions peut contenir des conditions

La boucle « for » (exemple)

Somme totale des valeurs comprises entre 1 et **n** (inclus) et somme des valeurs paires dans le même intervalle

```
# somme_paire.py - D:\Travaux\university\Cours_Univer...
File Edit Format Run Options Window Help
# -*- coding: utf -*-
#valeur limite
n = int(input("n : "))

#initialisation
sommePaire = 0
sommeTotale = 0

#effectuer la somme
for i in range(1,n+1):
    #somme si valeur paire
    if (i % 2 == 0):
        sommePaire = sommePaire + i
    #on n'est plus dans le " if " ici
    #mais on est bien dans le " for "
    sommeTotale = sommeTotale + i

#on est sorti du " for " ici
#affichage avec transptypage
print("somme des valeurs paires : " + str(sommePaire))
print("somme totale : " + str(sommeTotale))

#pour bloquer la fermeture de la console
input("pause...")
```

Il faut mettre **n+1** dans range() pour que **n** soit inclus dans la somme

Observez attentivement les indentations.

Structures de données

Accès aux caractères individuels d'une chaîne

Chaînes de caractères <=> Données composites

Dans le cas d'une chaîne de caractères, ses entités plus simples sont évidemment les caractères eux-mêmes.

Python est pourvu de mécanismes qui permettent d'accéder séparément à chacun des caractères d'une chaîne

Exemple :

```
>>> ch = "Stéphanie"  
>>> print ch[0], ch[3]  
S p
```

Opérations élémentaires sur les chaînes

De nombreuses ***fonctions intégrées à Python***, permettent d'effectuer divers traitements sur les chaînes de caractères.

concaténation

```
a = 'Petit poisson'
b = ' deviendra grand'
c = a + b
print( c)
petit poisson deviendra grand
```

len()

```
>>> print len(c)
29
```

Conversion en nombre d'une chaîne de caractères qui représente un nombre

```
>>> ch = '8647'
>>> print ch + 45
==> *** erreur ***
>>> n = int(ch)      #on ne peut pas additionner une chaîne et un nombre
>>> print n + 65
8712                  # OK : on peut additionner 2 nombres
```

Exercices – Les chaines

- Écrivez un script qui détermine si une chaîne contient ou non le caractère «e».
- Écrivez un script qui compte le nombre d'occurrences du caractère «e» dans une chaîne.
- Écrivez un script qui recopie une chaîne dans une nouvelle variable, en insérant des astérisques entre les caractères. Ainsi par exemple, « gaston » devra devenir « g*a*s*t*o*n »
- Écrivez un script qui recopie une chaîne (dans une nouvelle variable) en l'inversant. Ainsi par exemple, « abcrib » deviendra « bircba ».
- En partant de l'exercice précédent, écrivez un script qui détermine si une chaîne de caractères donnée est un palindrome (c'est-à-dire une chaîne qui peut se lire indifféremment dans les deux sens), comme par exemple « radar » ou « s.o.s ».