

Programmation Python: Les bases de Python



Pr. BENDAHDANE

Types de données, variables, opérations

BASES DE PYTHON

Les Variables

- Une **variable** est un nom placé à un endroit de la mémoire où un programmeur peut stocker des données et récupérer ensuite les données en utilisant le "nom" de la **variable**
- Il appartient aux programmeurs de choisir le nom des **variables**
- Il est possible de changer le contenu d'une **variable** lors d'une affectation ultérieure

x = 12.2

y = 14

x 12.2

y 14

Les Variables

- Une **variable** est un nom placé à un endroit de la mémoire où un programmeur peut stocker des données et récupérer ensuite les données en utilisant le "nom" de la **variable**
- Il appartient aux programmeurs de choisir le nom des **variables**
- Il est possible de changer le contenu d'une **variable** lors d'une affectation ultérieure

x = 12.2

y = 14

x = 100

x ~~12.2~~ 100

y 14

Les règles de nommage des Variables Python

- Commencent obligatoirement par une lettre ou un souligné _
- Contiennent des lettres, des nombres et des soulignés
- Sont sensibles à la case Majuscule-Minuscule
- **Bon:** spam eggs spam23 _speed
- **Mauvais:** 23spam #sign var.12
- **Divers:** spam Spam SPAM

Mots réservés

- Vous ne pouvez pas prendre les **mots réservés** comme noms de variables / identifiants

and del for is raise assert
elif from lambda return
break else global not try
class except if or while
continue exec import pass
yield def finally in print as
with

Premières opérations

Affectation – Typage automatique

➤ `a = 1.2`

a est une variable, en interne elle a été automatiquement typée en flottant « float » parce qu'il y a un point décimal. **a** est l'identifiant de la variable (attention à ne pas utiliser les mots réservés comme identifiant), **=** est l'opérateur d'affectation

Calcul

➤ `d = a + 3`

d sera un réel contenant la valeur 4.2

Forcer le typage d'une variable (sert aussi pour le transtypage)

➤ `b = float(1)`

Même sans point décimal, **b** sera considéré comme float (**b = 1**, il aurait été int dans ce cas).

Connaître le type d'un objet

➤ `type(nom_de_variable)`

Affiche le type interne d'une variable (ex. `type(a)` → `<class 'float'>`)

Supprimer un objet de la mémoire

➤ `del nom_de_variable`

où `nom_de_variable` représente le nom de l'objet à supprimer.

Types élémentaires de Python

Types élémentaires de Python

- Numérique qui peut être **int** (entier) ou **float** (double). Les opérateurs applicables sont : `+`, `-`, `*`, `/` (division réelle), `**` (puissance), `%` (modulo), `//` (division entière)

- **bool** correspond au type booléen, il prend deux valeurs possibles **True** et **False** (respecter la casse). Les opérateurs sont **not** (négation), **and** (ET logique), **or** (OU logique)

ex. `not(True)` → `False` ; `True and False` → `False` ; etc.

- **str** désigner les chaînes de caractères. Une constante chaîne de caractère doit être délimitée par des guillemets (ou des quotes)

ex. `a ← « tano »` affecte la valeur « tano » à l'objet **a** qui devient donc une variable de type chaîne de caractères. Une chaîne de caractère se comporte comme un vecteur : `len()` pour connaître sa longueur, `a[0]` → « t », `a[1:3]` → « ano », `a[2:]` → « no », etc.

- Remarque : pour connaître la classe d'un objet i.e. le type associé à un objet, on utilise la fonction **type(nom_objet)**

ex. `type(1.2)` → renvoie la valeur 'float'

Instanciation et affectation

Affectation simple

La seconde évite les ambiguïtés.

```
#typage automatique
a = 1.0
#typage explicite
a = float(1)
```

Affectations multiples

Pas fondamental

```
#même valeur pour plusieurs variables
a = b = 2.5

#affectations parallèles
a, b = 2.5, 3.2
```

Opérations, expressions, enchaînements

La plus couramment utilisée

1 instruction = 1 ligne

```
a = 1
b = 5
d = a + b
```

Autres possibilités

Personne n'utilise ces écritures

```
a = 1;b = 5 ;d = a + b;
```

```
a = 1;
b = 5;
d = a + b;
```

Une opération particulière

Une variable ne se comporte pas de la même manière de part et d'autre du symbole d'affectation

```
a = 2
a = a + 1
```

Variables

EXEMPLE

script.py

```
1 myint = 7
2 print(myint)
```

script.py

```
1 myfloat = 7.0
2 print(myfloat)
3 myfloat = float(7)
4 print(myfloat)
```

script.py

```
1 mystring = 'hello'
2 print(mystring)
3 mystring = "hello"
4 print(mystring)
```

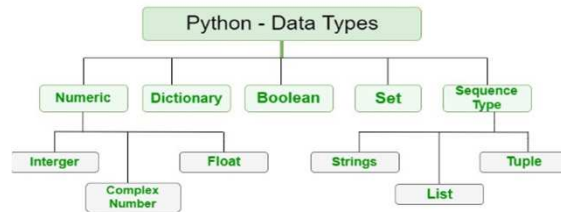
script.py

```
1 mystring = "Don't worry about apostrophes"
2 print(mystring)
```

Remarque : pour connaître le type d'une variable, on peut utiliser la fonction : `type(nomvar)`

script.py

```
1 one = 1
2 two = 2
3 three = one + two
4 print(three)
5
6 hello = "hello"
7 world = "world"
8 helloworld = hello + " " + world
9 print(helloworld)
```



11

Variables

TYPE BOOLEENNE & OPERATEUR DE COMPARAISON - LOGIQUE

- ☐ Pour un type « **bool** », les deux valeurs possibles **False (faux)**, **True (vrai)**.
- ☐ Les opérations de **comparaison** produisent un « bool » : `==`, `!=`, `>`, `<`, `>=`, `<=`
- ☐ Les opérateurs de logique : **and**, **or** et **not**

EXEMPLE

Opérateur unaire not

a	not(a)
False	True
True	False

Opérateurs binaires or et and

a	b	a or b	a and b
False	False	False	False
False	True	True	False
True	False	True	False
True	True	True	True

```
>>> 2 > 8
```

```
False
```

```
>>> 2 <= 8 < 15
```

```
True
```

```
>>> (3 == 3) or (9 > 24)
```

```
True
```

```
>>> (9 > 24) and (3 == 3)
```

```
False
```

```
>>> 'a' or False # 'a' est évalué à vrai
```

```
'a'
```

```
>>> 0 or 56 # 0 et 56 sont transtypés en booléen. 0 vaut False et les autres valeurs True
```

```
56
```

```
>>> b = 0
```

```
>>> b and 3>2 # b est transtypé en booléen
```

```
0
```

12

Variables

CONVERSION OU TRANSTYPAGE ou CASTING

❑ le passage d'un type à l'autre en python est possible, avec les fonctions : **int()**, **float()**, **bool()** et **str()** .

❑ Conversion automatique en **False** de :

- la valeur « Zero » (quel que soit le type numérique)
- les chaînes et conteneurs vides
- la constante **None**

❑ Toutes les autres valeurs sont converties en booléen vers **True**

EXEMPLE

```
>>> i = 3
>>> str(i)
'3'
>>> i = '456'
>>> int(i)
456
>>> float(i)
456.0
>>> i = '3.1416'
>>> float(i)
3.1416
```

```
In [3]: a = 0
In [4]: bool(a)
...:
Out[4]: False
In [5]: a = 1
In [6]: bool(a)
Out[6]: True
In [7]: bool(2)
Out[7]: True
In [8]: a = None
In [9]: bool(a)
Out[9]: False
```

13

Transtypage

Principe

Utilisation du mot-clé désignant le type

> **nouveau_type** (objet)

Conversion en numérique

a = « 12 » # a est de type chaîne caractère

b = float(a) # b est de type float

N.B. Si la conversion n'est pas possible ex. float(« toto »), Python renvoie une erreur

Conversion en logique

a = bool(« TRUE ») # a est de type bool est contient la valeur True

a = bool(1) # renvoie True également

Conversion en chaîne de caractères

a = str(15) # a est de type chaîne et contient « 15 »

Expressions Numériques

```
>>> xx = 2
>>> xx = xx + 2
>>> print xx
4
>>> yy = 440 * 12
>>> print yy
5280
>>> zz = yy /
1000
>>> print zz
5
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print kk
3
>>> print 4 **
3
64
```

Nom	Désignation	Exemple	Résultat
+	Addition	34+1	35
-	Soustraction	34.0-0.1	33.9
*	Multiplication	300*30	9000
/	Division	1/2	0.5
//	Division entière	1//2	0
**	Exponentiation	4**0.5	2.0
%	Modulo	20%3	2

Opérateur	Opération
+	Addition
-	Soustraction
*	Multiplication
/	Division
**	Puissance
%	Reste

Ordre d'évaluation

- Quand nous chaînons les opérateurs ensemble - Python doit connaître leur ordre de traitement
- Ceci est appelé “**priorité des opérateurs**”
- Quel est l'opérateur qui “a une priorité” sur les autres ?

x = 1 + 2 * 3 - 4 / 5 ** 6

Règles de priorité des opérateurs

Règle de la priorité la plus haute à la plus basse:

- > Les parenthèses sont toujours respectées
- > L'exposant (élévation à la puissance)
- > Multiplication, division, et reste
- > Addition et soustraction
- > De gauche à droite

Parenthèses
Puissance
Multiplication
Addition
De gauche à droite

Priority	Operator	
1	<code>+, -</code>	unary
2	<code>**</code>	
3	<code>*, /, //, %</code>	
4	<code>+, -</code>	binary

```
>>> x = 1 + 2 ** 3 / 4 * 5
>>> print x
11
>>>
```

1 + 2 ** 3 / 4 * 5

1 + 8 / 4 * 5

1 + 2 * 5

1 + 10

11

Parenthèses
Puissance
Multiplication
Addition
De gauche à droite



PYTHON



Saisie et affichage à la console

ENTRÉES ET SORTIES

ENTREE/SORTIE

DEFINITION

Les échanges de l'utilisateur avec le programme, en mode « console » se fait par la *saisie des informations **entrée***, généralement depuis **une lecture au clavier**. Inversement, on *affiche ou on fait **sortir*** des informations par une écriture sur **l'écran**.

Input
Output

EN PYTHON :

- ☐ la saisie se fait avec la fonction « **input()** » et l'affichage à l'écran se fait avec la fonction « **print()** ».
- ☐ la fonction de saisie « input() » retourne **un résultat en format « texte »** et il peut contenir un **message d'invite de saisie**
- ☐ la fonction d'affichage « print() » affiche à la fois du texte et les valeurs des variables sans une **obligation** de précision des formats (**l'affichage formaté**)

ENTREE/SORTIE

EXEMPLE

Input

```
>>> f = input("Entrez un flottant : ")
Entrez un flottant : 12.345
>>> type(f)
str
>>> g = float(f) # transtypage
>>> type(g)
float
```

```
>>> i = input("Entrez un entier : ")
Entrez un entier : 3
>>> i
'3'
>>> iplus = int(input("Entrez un entier : ")) + 1
Entrez un entier : 3
>>> iplus
4
>>> ibug = input("Entrez un entier : ") + 1
Entrez un entier : 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
```

Print

```
>>> print('Hello World!')
Hello World!
>>> print() # affiche une ligne blanche

>>> a, b = 2, 5
>>> print('Somme : ', a + b, ', ', a - b, 'est la différence et', a * b, 'le produit.')
Somme : 7 ; -3 est la différence et 10 le produit.
>>> :
>>> print(a, b, sep='***', end='@') # utilise un séparateur et une fin de ligne spécifiques
2***5@
```

21

Print

Code	Console Output
1 print() #ligne vide , saut de ligne	
2 print("bonjour python")	bonjour python
3 print('bonjour python')	bonjour python
4 print("bonjour "python")	bonjour "python"
5 print('bonjour 'python')	bonjour 'python'
6 print("bonjour \"python\"")	bonjour "python"
7 print("bonjour", "python")	bonjour python
8 print("bonjour", "pyhton", end='\n')	bonjour pyhton
9 print("abc")	abc
10 print("xyz")	xyz
11 print("abc", end='@')	abc@xyz
12 print("xyz")	xyz
13 print("ab", "cd", "ef", sep='-')	ab-cd-ef
14	

Keyword argument : fin de ligne spécifique (par défaut \n)

Print('abc', end='@')

Print('ab', 'cd', sep='-')

Keyword argument : séparateur

Print

```

1 print(0o10) # convertir 10 octale en decimal
2 print(0xA1) # convertir A1 Hexadecimal en decimal
3
Console >_
8
161

```

input

Saisie

```

a = input (« Saisir votre valeur »)
a = float(a)

```

`input()` permet d'effectuer une saisie au clavier

Il est possible d'afficher un message d'invite

La fonction renvoie toujours une chaîne, il faut convertir

Affichage

```

#Affichage explicite
print(a)

```

- Un affichage multiple est possible

Ex. `print(a,b)` #affiche a et b

- L'affichage direct du contenu d'un tableau (liste) est possible également.

input

```

1 x = input("entrez un entier : ")
2 x=float(x)
3
4 # 2ème solution
5
6 x=int(input("entrez un entier : "))
7

```

Console >_

```

entrez un entier : 6
entrez un entier : 6

```

Convertir en int : int(x)
 Convertir en float : float(x)
 Convertir en string : str(x)

Un exemple

```

File Edit Format Run Options Window Help
# -*- coding: utf -*-

#saisie à la console d'un prix HT
pht = input("prix HT : ")

#transtypage en numérique
pht = float(pht)

#calcul TTC
pttc = pht * 1.20

#affichage avec transtypage
print("prix TTC : " + str(pttc))

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

```

Ln: 1 Col: 0

Pour gérer correctement
l'affichage des caractères
accentués

print("« prix ttc : », pttc)
Fonctionne également


Concaténation de 2 chaînes de caractères

Branchements conditionnels et boucles

STRUCTURES ALGORITHMIQUES

Branchement conditionnel « if »

Condition est très souvent une
opération de comparaison



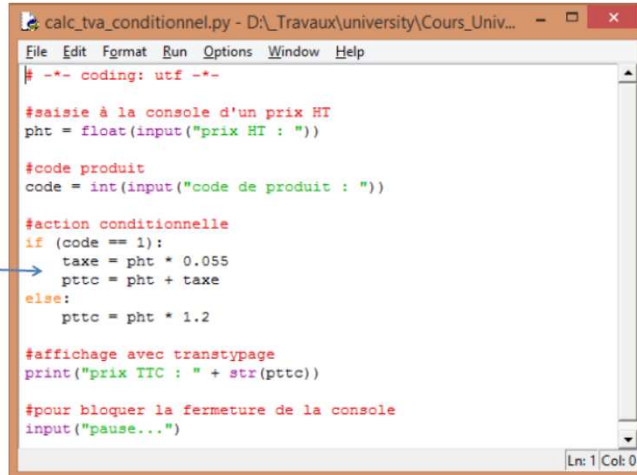
```
if condition:
    bloc d'instructions
else:
    bloc d'instructions
```

- (1) Attention au **:** qui est primordial
- (2) C'est l'**indentation** (le décalage par rapport à la marge gauche) qui délimite le bloc d'instructions
- (3) La partie **else** est facultative

Branchement conditionnel « if » (exemple)

Noter l'imbrication des blocs.

Le code appartenant au même bloc doit être impérativement aligné sinon erreur.



```

calc_tva_conditionnel.py - D:\Travaux\university\Cours_Univ...
File Edit Format Run Options Window Help
# -*- coding: utf-8 -*-

#saisie à la console d'un prix HT
pht = float(input("prix HT : "))

#code produit
code = int(input("code de produit : "))

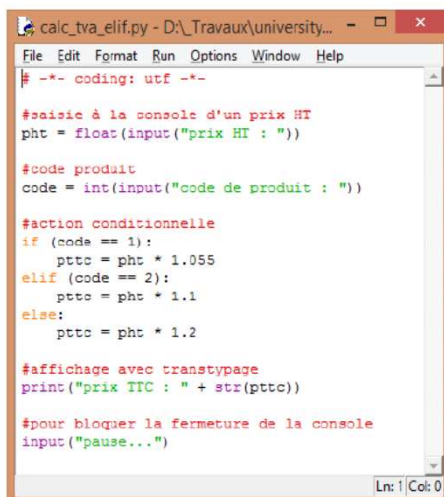
#action conditionnelle
if (code == 1):
    taxe = pht * 0.055
    pttc = pht + taxe
else:
    pttc = pht * 1.2

#affichage avec transtypage
print("prix TTC : " + str(pttc))

#pour bloquer la fermeture de la console
input("pause...")
Ln: 1 Col: 0

```

Succession de if avec elif



```

calc_tva_elif.py - D:\Travaux\university...
File Edit Format Run Options Window Help
# -*- coding: utf-8 -*-

#saisie à la console d'un prix HT
pht = float(input("prix HT : "))

#code produit
code = int(input("code de produit : "))

#action conditionnelle
if (code == 1):
    pttc = pht * 1.055
elif (code == 2):
    pttc = pht * 1.1
else:
    pttc = pht * 1.2

#affichage avec transtypage
print("prix TTC : " + str(pttc))

#pour bloquer la fermeture de la console
input("pause...")
Ln: 1 Col: 0

```

- **elif** n'est déclenché que si la (les) condition(s) précédente(s) a (ont) échoué.
- **elif** est situé au même niveau que **if** et **else**
- On peut en mettre autant que l'on veut



Il n'y a pas de `switch()` ou de `case...of` en Python

Avant la boucle « for » : génération d'une séquence de valeurs

Principe de la boucle for

Elle ne s'applique que sur une collection de valeurs. Ex. tuples, listes,... à voir plus tard.

Suite arithmétique simple (séquence de valeurs entières)

On peut définir des boucles indicées en générant une collection de valeurs avec `range()`

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

Un pas de 2

Boucle « for »

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

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

Remarque :

- 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

Boucle « for » (exemple)

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

```

# -*- coding: utf-8 -*-

# 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 + 1

# on est sorti du "for" ici
# affichage avec transtypage
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.

Boucle For

FOR : parcourir

La condition de début et de fin de la boucle sont connu d'avance. C'est pour parcourir un itérable tour à tour, pour accéder à chaque valeur afin de la traiter dans le corps de la boucle

EXEMPLE

```

>>> for lettre in "ciao": # On peut itérer sur les caractères des chaînes
...     print(lettre, lettre.upper())
...
c C
i I
a A
o O

>>> for x in [2, 'a', 3.14]: # Notation pour une liste de valeurs (cf. chap. 4)
...     # Le typage dynamique de Python permet à x de recevoir à
...     # chaque itération une valeur d'un type différent
...     print(x, 2*x)

>>> for i in range(6): # Générateur de séquences d'entiers à la demande
...     print(i, i ** 2)
...
0 0
1 1
2 4
3 9
4 16
5 25

>>> nb_voyelles = 0
>>> for lettre in "Python est un langage fort sympa":
...     if lettre.lower() in "aeiouy":
...         nb_voyelles = nb_voyelles + 1
...
>>> nb_voyelles
10
  
```

34

STRUCTURE DE CONTROLE

BREAK – CONTINUE

- ☐ L'interruption de la boucle se fait avec l'instruction **BREAK**
- ☐ Pour court-circuiter une boucle, on utilise **CONTINUE** (passer à l'itération suivante sans continuer les instructions de la boucle)

EXEMPLE

```
for x in range(1, 11):
    if x == 5:
        break
    print(x, end=" ") # end=" " remplace le retour à la ligne du print() par un simple espace
print("Boucle interrompue pour x=", x)
```

```
1 2 3 4 Boucle interrompue pour x= 5
```

```
>>> for x in range(1, 11):
...     if x == 5:
...         continue
...     print(x, end=" ")
...
1 2 3 4 6 7 8 9 10
>>> # la boucle a sauté la valeur 5
```

35

La clause **else** de la Boucle for

- La clause **else** dans un boucle permet de définir un bloc d'instructions qui sera exécuté à la fin seulement si la boucle s'est déroulée complètement sans être interrompue par un **break**.

```
for x in range(6):
    print(x)
else:
    print("Finally finished!")
```

```
0
1
2
3
4
5
Finally finished!
```

```
for n in range(2, 8):
    for x in range(2, n):
        if n % x == 0:
            print(n, "egale", x, "*", n/x)
            break
    else:
        print(n, "est un nombre premier")
```

```
2 est un nombre premier
3 est un nombre premier
4 egale 2 * 2.0
5 est un nombre premier
6 egale 2 * 3.0
7 est un nombre premier
```

Boucle while

Opération de comparaison
Attention à la boucle infinie !

while condition:
 bloc d'instructions

Remarque :

- Attention à l'indentation toujours
- On peut « casser » la boucle avec **break**

Boucle « while » (exemple)

```

somme_paire_while.py - D:/Travaux/university/Co...
File Edit Format Run Options Window Help
# -*- coding: utf-8 -*-

#valeur limite
n = int(input("n : "))

#initialisation
sommePaire = 0
sommeTotale = 0

#effectuer la somme
i = 1
while (i <= n):
    #somme si valeur paire
    if (i % 2 == 0):
        sommePaire = sommePaire + i
    #somme toujours, paire ou pas
    sommeTotale = sommeTotale + 1
    #incrémenter
    i = i + 1

#affichage avec transtypage
print("somme des valeurs paires : " + str(sommePaire))
print("somme totale : " + str(sommeTotale))

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

```

Ne pas oublier
l'initialisation de **i**

Observez attentivement
les indentations.

L'équivalent de la Boucle Répéter jusqu'à

- La boucle répéter n'existe pas en python, on peut utiliser l'astuce suivante :

```
Répéter
# ton code
Jusqu'à condition
```

```
1 while True:
2     # Ton code...
3     if condition:
4         break
```

La clause **else** de la Boucle while

- Avec l'instruction **else**, nous pouvons exécuter un bloc de code **une fois** lorsque la condition **n'est plus vraie** :

```
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")
```

```
1
2
3
4
5
i is no longer less than 6
~
```

Boucle while

□ **WHILE** : Répéter les instructions du bloc, sachant que la condition à la boucle d'accès (répétitions des instruction) est connu au préalable.

EXEMPLE

```
>>> x, cpt = 257, 0
>>> sauve = x
>>> while x > 1:
...     x = x // 2      # division avec troncature
...     cpt = cpt + 1  # incrémentation
...
>>> print("Approximation de log2 de", sauve, ":", cpt)
Approximation de log2 de 257 : 8

n = int(input('Entrez un entier [1 .. 10] : '))
while not(1 <= n <= 10):
    n = int(input('Entrez un entier [1 .. 10], S.V.P. : '))
```

```
saisie_ok = False
while not saisie_ok : # tant que saisie_ok est False, faire :
    a = int(input("Entrez un nombre entier de 1 à 100 divisible par 3 : "))
    b = int(input("Entrez un nombre entier pair supérieur à " + str(a) + " : "))
    saisie_ok = (1 <= a <= 100) and (b % 2 == 0) and (a < b)
```

```
import random

choix = random.randint(0, 100)

while True:
    reponse = int(input('Devinez le nombre: '))
    if reponse < choix:
        print('Plus grand')
    elif reponse > choix:
        print('Plus petit')
    else:
        break
print('Bravo')
```

41

(*) L'alternative de la boucle
DO...WHILE



Travaux dirigés



Exercices

Exercice 0

Ecrire un algorithme qui échange la valeur de deux variables. Exemple, si $a = 2$ et $b = 5$, le programme donnera $a = 5$ et $b = 2$.

Exercice 1

Ecrire un algorithme qui demande un nombre à l'utilisateur, puis qui calcule et affiche le carré de ce nombre.

Exercice 2

Ecrire un algorithme qui lit le prix HT d'un article, le nombre d'articles et le taux de TVA, et qui fournit le prix total TTC correspondant. Faire en sorte que des libelles apparaissent clairement.

Exercice 3

Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de côté le cas où le produit est nul). Attention toutefois : on ne doit pas calculer le produit des deux nombres.

Exercice 4

Ecrire un algorithme qui demande l'âge d'un enfant à l'utilisateur. Ensuite, il l'informe de sa catégorie :

"Poussin" de 6 à 7 ans

"Pupille" de 8 à 9 ans

"Minime" de 10 à 11 ans

"Cadet" après 12 ans

Exercice 5

Ecrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : Plus petit ! , et inversement, Plus grand ! Si le nombre est inférieur à 10.

Exercice 6

Ecrire un algorithme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre.

Exercice 7

Ecrire un algorithme qui demande un nombre de départ, et qui calcule sa factorielle.

Exercice 8

écrivez un algorithme permettant, à l'utilisateur de saisir les notes d'une classe. L'algorithme, une fois la saisie terminée, renvoie le nombre de ces notes supérieures à la moyenne de la classe.

43