www.dimension-k.com

Activité : découverte de Python

Symbole	action	exemple	effet
+, -, *, /	Quatre opérations	5-3*4+3/2	5.5
//	Quotient de la division euclidienne	13//4	3
%	Reste de la division euclidienne	13%4	1
**	puissance	2**3	8
abs()	Valeur absolue	abs(5-11)	6
sqrt()	Racine carrée d'une valeur*	sqrt(196)	14

Variable

Quand on veut retenir une valeur pour pouvoir l'utiliser plus tard, on la stocke en mémoire dans une variable. Exemple : *ordonnee = -14* si plus tard on tape abs(ordonnée)*2-5 l'ordinateur nous répondra 23.

Sortie

Si on veut qu'un programme affiche dans la console des éléments de texte ou des résultats, on utilisera la fonction *print(élément1, élément2, …).*
Astuces intéressantes dans la correction de la mise en pratique 1.

Entrée

Pour obtenir une information de la part de l'utilisateur on utilise la commande input. Exemple : *abscisseA = float(input(" donner l'abscisse du point A"))*
Récupérera l'information donnée et la stockera dans la variable abscisseA.

Mise en pratique 1 :

Ecrire un programme demandant les coordonnées de deux points A et B et les stockant dans les variables : abscisseA, ordonneeA, …

1) Le programme donnera alors les coordonnées abscisseI et ordonneeI du milieu I du segment $[AB]$
2) Il donnera aussi le carré de la longueur AB.
3) Enregistrer votre travail sous le nom « introduction-ex1.py » dans le répertoire maths situé dans le répertoire document.

Commentaires :

Pour aider les personnes qui vont relire nos programmes, on peut clarifier notre code en rajoutant en fin de ligne le symbole # suivi de notre explication. Python ignorera ce qui suit le #.

Exemple (donnant une information importante sur la dernière ligne du tableau*) :

```
AB=sqrt(valeur)  #la fonction sqrt a été importée à la première ligne du
                 # programme avec la commande « from math import sqrt »
```

Structure conditionnelle

Notre programme doit pouvoir s'adapter à toute sorte de situation.
Télécharger le fichier structureNote.py sur le site ou recopie le script suivant dans un nouveau programme (tu peux copier/coller la version numérique sur le site)

```
note=float(input("quelle est ta note sur 20 ?"))
if note>10 :          #.....
   diff = note-10     #.....
   print("tu as eu ",diff,"points audessus de la moyenne")
elif note==10 :       #.....
   print("tu as eu exactement la moyenne")
else :                #.....
   diff = 10 - note   #.....
   print("il te manque ",diff," points pour avoir la moyenne")
assure toi que les décalages soient bien des tabulations et non des espaces
```

Commenter les lignes où il y a le symbole #

Remarque : les lignes elif et else sont facultatives

Fonctions

On peut créer nous-même nos propres fonctions :

Règle générale

```
def nomFonction(parametre1,…) :
        " " " documentation de la
        fonction : qu'utilise t elle
        Que donne-t-elle ? " " "
        Actions et calcul
        sortie de l'information
```

Exemple

```
def distance(val1,val2) :
        " " " calcule la distance entre les
        points d'abscisse val1 et val2 " " "

        dist = abs(val1-val2)
        return dist
```

Une fois le programme contenant la fonction lancée, celle-ci sera chargée en mémoire et utilisable n'importe quand dans la console. Je pourrai alors taper « *distance(23,-15)* » et l'ordinateur me répondra « 38 »

Attention : toutes les lignes de codes servant à programmer la fonction sont décalées d'une tabulation par rapport à la première ligne de celle-ci.

Remarque : dans un programme les fonctions n'ont pas de sortie console, si vous voulez avoir un affichage du résultat de vos actions il faudra utiliser la fonction print. Par exemple : « print("la distance entre 23 et -15 est ", distance(23,-15)) »

Mise en pratique 2 :

Créer un nouveau programme dans lequel vous écrirez les fonctions suivantes :

1) La fonction **f** de paramètre x, qui renvoie la valeur de $x^2 - x - 5$
2) La fonction **g** de paramètre x, qui renvoie la valeur de $-2x + 5$
3) La fonction **audessus** de paramètre x, qui renvoie « f » si $f(x) > g(x)$, « g » si $f(x) < g(x)$ et « égalité » sinon.

Remarque : la fonction audessus va donc utiliser les fonctions définies aux questions 1 et 2.

Mise en pratique 3 :

Créer un nouveau programme dans lequel vous écrirez les fonctions suivantes :

1) La fonction **distanceC** de paramètres x1, y1, x2, y2 qui renvoie le carré de la distance entre les points dont on vient de donner les coordonnées.
2) La fonction rectangle de paramètres xA,yA, xB, yB, xC, yC qui renvoie « A » si ABC est rectangle en A, « B » si ABC est rectangle en B, … , « nulle part » sinon.

Mise en pratique 4 :

Créer un nouveau programme dans lequel vous écrirez les fonctions suivantes :

1) La fonction **milieu** de paramètres xM, yM, xN, yN (coordonnées des points M et N, qui renvoie les coordonnées de I le milieu du segment [MN].
2) La fonction **parallelogramme** de paramètres xA, yA, xB, …les coordonnées de A, B, C et D les sommets d'un quadrilatère et qui renvoie « parallélogramme » ou « quelconque » selon la nature du quadrilatère.
3) La fonction **symetrique** de paramètres xA, yA, xI et yI les coordonnées de A et de I, et qui renverra xB et yB les coordonnées de B le symétrique de A par rapport à I. (autrement dit le point B tel que I est le milieu de [AB])

Remarque :
Quand on veut sortir plusieurs valeurs d'une fonction on les sépare par des « , ».
Par exemple la fin de la fonction de la question 1 pourrait être : « return xI,yI »

La boucle while

On peut être amené à répéter un grand nombre de fois la même séquence d'actions. Dans ce cas on utilisera une boucle while ou une boucle for.
Télécharger le fichier tableauValeurs.py sur le site ou recopie le script :

```
def h(x):
    return 0.5*x**2+3*x-4

x=valdep=4  #on donne à x et à valdep la valeur 4
p = 0.5   #
while x<=14 : #
    print(x,"a pour image",h(x))
    x=x+p #
```

Assure-toi que les décalages soient bien des tabulations et non des espaces

1) Commenter les lignes où il y a le symbole #
2) Comment modifier le programme pour qu'il donne les images des x allant de -10 à 10 (avec un pas de 0,2) ? appliquer ce changement.

Mise en pratique 5 :

Créer un nouveau programme permettant de chercher (maladroitement) la valeur du minimum de la fonction $g: x \to x^2 - 10x + 21$ sur un intervalle donné. On va créer une fonction miniG de paramètres Xmin, Xmax, Pas qui renverra la plus petite image obtenue sur l'intervalle quand on le parcourt en avançant d'un Pas à chaque fois.
Conseil : avant la boucle on pourra fixer minimum = g(Xmin), et à chaque tour de boucle on comparera minimum avec l'image obtenue et on remplacera minimum par la valeur trouvée si elle est plus petite.

Mise en pratique 6 :

1) Créer la fonction diviseur de paramètres *a* et *m* qui renvoie *True*, si *a* est divisible par *m* et *False* sinon.

Supposons que l'on a une fraction $\frac{num}{denom}$ et qu'on veut la simplifier par k.

2) Créer un nouveau programme contenant une fonction simpli(num, deno, k) qui renverra les nouveaux numérateur et dénominateurs si on peut simplifier par k et les anciens dans le cas contraire.

Mise en pratique 7 :

Créer un nouveau programme contenant une fonction PGCD de paramètres *a* et *b* qui déterminera le plus grand diviseur commun de *a* et *b.*

Astuces / conseils

Mise en pratique 4

Vu qu'on a créé une fonction spécialement pour déterminer les coordonnées du milieu d'un segment à la question 1, la méthode parmi de nombreuses autres qui nous permettra à la question 2 de savoir si un quadrilatère est un parallélogramme va de soi.

Pour vérifier si les coordonnées des milieux de $[MN]$ et $[RS]$ sont égales on peut écrire : « *if milieu(xM,yM,xN,yN)==milieu(xR,yR,xS,yS) :* »

Pour la question 3)

On a déjà vu en exercice et en évaluation comment trouver les coordonnées du symétrique d'un point connu par rapport à un autre point connu.

A titre de rappel, dans l'évaluation sujet A on avait à peu de choses près :

D symétrique de $B(-3;-1)$ par rapport à $M(4;5{,}5)$ ⇔M est le milieu $[BD]$

$$\Leftrightarrow M(4;5{,}5) = M\left(\frac{x_B+x_D}{2};\frac{y_B+y_D}{2}\right) \Leftrightarrow \begin{cases} 4 = \frac{-3+x_D}{2} \\ 5{,}5 = \frac{-1+y_D}{2} \end{cases} \Leftrightarrow \begin{cases} 4\times 2 = -3 + x_D \\ 5{,}5\times 2 = -1 + y_D \end{cases}$$

$$\Leftrightarrow \begin{cases} 8 + 3 = x_D \\ 11 + 1 = y_D \end{cases} \Leftrightarrow D(11;12)$$

Il vous faudra adapter cette démonstration à notre situation, la résoudre sur le papier et grâce aux formules obtenues écrire la fonction **symetrique**.

La démonstration commence ainsi :

B symétrique de $A(x_A;x_B)$ par rapport à $I(x_I;y_I)$ ⇔I est le milieu $[AB]$

$\Leftrightarrow I(x_I;y_I) = I\left(\frac{x_A+x_B}{2};\ldots\right)$

Et ça se termine par $B(2x_I - x_A;\ldots)$

Correction colorée

Le code est disponible plus loin dans le document pour faire du copier / coller

Mise en pratique 1

```
abscisseA=float(input("abscisse du point A : "))
ordonneeA=float(input("ordonnée du point A : "))
abscisseB=float(input("abscisse du point B : "))
ordonneeB=float(input("ordonnée du point B : "))

abscisseI=(abscisseA+abscisseB)/2
ordonneeI=(ordonneeA+ordonneeB)/2

print("I a pour coordonnées(",abscisseI,",",ordonneeI,")")
#cette écriture est un peu lourde à chaque fois qu'on introduit
#la valeur d'une variable on doit interompre la chaine avec " puis
#une virgule
print("I a pour coordonnées({},{})".format(abscisseI,ordonneeI))
#avec cette présentation on utilisera {} pour dire qu'on va insérer
#le contenu d'une variable. Dans la partie format on indiquera les
#différentes variables utilisées dans leur ordre d'utilisation.
print(f"I a pour coordonnées({abscisseI},{ordonneeI})")
#simplifiaction de l'approche précédente. L'indication de formatage se
#fait avec la lettre f collée avant les guillemets.
#les nom de variables sont cités directement dans les accolades.
print(f"AB²={(abscisseB-abscisseA)**2+(ordonneeB-ordonneeA)**2}")
#on peut mettre des calculs directement dans les accolades, print
#affichera leur résultat.
```

Mise en pratique 2

```
def distance(val1,val2):
  return abs(val1-val2) #abs() enlève le signe de la différence

print(distance(-7,23)) #petit essai, le print est obligatoire sinon
#il n'y aura pas d'affichage

def f(x):
  return x**2-x-5 #retour du résultat du calcul, pas besoin de varible

def g(x):
  return -2*x+5

def audessus(x):
  if f(x)>g(x):
    return "f"
  elif f(x)<g(x):
    return "g"
  else:
    return "égalité"
x=-3
print(f"f({x})={f(x)} et g({x})={g(x)}")
print(f"pour {x} la fonction au dessus est {audessus(x)}.")
```

Mise en pratique 3

```
def distanceC(x1,y1,x2,y2):
  return (x2-x1)**2+(y2-y1)**2

print("le carré de la distance entre A et B sera : ")
print(distanceC(5,10,3,-19))

def rectangle(xA,yA,xB,yB,xC,yC):
  ABc=distanceC(xA,yA,xB,yB) #ABc récupère la sortie de distanceC
  BCc=distanceC(xB,yB,xC,yC) #ça serait impossible avec un print
  ACc=distanceC(xA,yA,xC,yC)
  if ABc==ACc+BCc :
    print("ABC est rectangle en C")
  elif ACc==BCc+ABc:
    print("ABC est rectangle en B")
  elif BCc==ACc+ABc:
    print("ABC est rectangle en A")
  else :
    print("le triangle n'est pas rectangle")

rectangle(-2,-2,3,-4,2,8) #pas de besoin de print, celui ci est déjà
#inclu dans la fonction
```

Mise en pratique 4

```

def milieu(xM,yM,xN,yN):
  xI=(xM+xN)/2
  yI=(yM+yN)/2
  return xI,yI

def parallelogramme(xA,yA,xB,yB,xC,yC,xD,yD):
  if milieu(xA,yA,xC,yC)==milieu(xB,yB,xD,yD):
    print("Le quadrilatère est un parallélogramme.")
  else :
    print("Le quadrilatère n'est pas un parallélogramme.")

parallelogramme(6,-2,9,-1,10,2,7,1) #test avec un parallélogramme
parallelogramme(6,-2,9,-1,10,2,7,-1)#test avec un quadri quelconque

def symetrique(xA,yA,xI,yI):
  xB=2*xI-xA #I étant le milieu de [AB] on a retravaillé les formules
  yB=2*yI-yA #du milieu d'un segment.
  return xB,yB
```

Mise en pratique 5

```python
def g(x):
  return x**2-10*x+21

def min(Xmin,Xmax,pas):
  minY=g(Xmin)
  x=Xmin
  xRef=x #première ligne d'un petit bonus non demdandé
  while x<Xmax:
    x=x+pas
    if minY>g(x):
      minY=g(x)
      xRef=x #seconde
  print(f"le minimum de g sur [{Xmin},{Xmax}] est y={minY}")
  print(f"il est atteint en x={xRef}.") #troisième

min(-20,20,0.1) "petit test : recherche du minimum entre -20 et 20.'
```

Mise en pratique 6

```python
def diviseur(a,m):
  """Entrées : a et m deux entiers
  Sortie : booléen qui indique si a est divisible par m"""
  #ces triples guillemets encadrent la documentation de la fonction
  #c'est facultatif mais ça aidera les gens qui voudront l'utiliser
  if a%m==0: #on regarde si le reste de la division de a
    return True  #par m vaut 0 (a divisible par m) ou pas
  else :
    return False

#version expresse
def diviseR(a,m):
  return a%m==0 #ce test a naturellement pour résultat True ou False

def simpli(num, deno, k):
  if diviseur(num,k) and diviseur (deno,k):
    print(f"la fraction vaut{num/k}/{deno/k} ")
  else :
    print(f"la fraction vaut{num}/{deno} ")
```

Mise en pratique 7

```python
def PGCD(a,b):
  """recherche le pgcd de a et b deux entiers non nuls"""
  dividende = a
  diviseur =b
  reste=1
  while True:
    quotient=dividende//diviseur
    reste=dividende%diviseur
    if reste==0 :
      return diviseur #à ce stade il vaut le dernier reste non nu
    dividende=diviseur
    diviseur = reste
```

Mise en pratique 1

```
abscisseA=float(input("abscisse du point A : "))
ordonneeA=float(input("ordonnée du point A : "))
abscisseB=float(input("abscisse du point B : "))
ordonneeB=float(input("ordonnée du point B : "))

abscisseI=(abscisseA+abscisseB)/2
ordonneeI=(ordonneeA+ordonneeB)/2

print("I a pour coordonnées(",abscisseI,",",ordonneeI,")")
#cette écriture est un peu lourde à chaque fois qu'on introduit
#la valeur d'une variable on doit interompre la chaine avec " puis
#une virgule
print("I a pour coordonnées({},{})".format(abscisseI,ordonneeI))
#avec cette présentation on utilisera {} pour dire qu'on va insérer
#le contenu d'une variable. Dans la partie format on indiquera les
#différentes variables utilisées dans leur ordre d'utilisation.
print(f"I a pour coordonnées({abscisseI},{ordonneeI})")
#simplifiaction de l'approche précédente. L'indication de formatage se
#fait avec la lettre f collée avant les guillemets.
#les nom de variables sont cités directement dans les accolades.
print(f"AB²={(abscisseB-abscisseA)**2+(ordonneeB-ordonneeA)**2}")
#on peut mettre des calculs directement dans les accolades, print
#affichera leur résultat.
```

Mise en pratique 2

```
def distance(val1,val2):
  return abs(val1-val2) #abs() enlève le signe de la différence

print(distance(-7,23)) #petit essai, le print est obligatoire sinon
#il n'y aura pas d'affichage

def f(x):
  return x**2-x-5 #retour du résultat du calcul, pas besoin de varible

def g(x):
  return -2*x+5

def audessus(x):
  if f(x)>g(x):
    return "f"
  elif f(x)<g(x):
    return "g"
  else:
    return "égalité"
x=-3
print(f"f({x})={f(x)} et g({x})={g(x)}")
print(f"pour {x} la fonction au dessus est {audessus(x)}.")
```

Mise en pratique 3

```
def distanceC(x1,y1,x2,y2):
  return (x2-x1)**2+(y2-y1)**2

print("le carré de la distance entre A et B sera : ")
print(distanceC(5,10,3,-19))

def rectangle(xA,yA,xB,yB,xC,yC):
  ABc=distanceC(xA,yA,xB,yB) #ABc récupère la sortie de distanceC
  BCc=distanceC(xB,yB,xC,yC) #ça serait impossible avec un print
  ACc=distanceC(xA,yA,xC,yC)
  if ABc==ACc+BCc :
    print("ABC est rectangle en C")
  elif ACc==BCc+ABc:
    print("ABC est rectangle en B")
  elif BCc==ACc+ABc:
    print("ABC est rectangle en A")
  else :
    print("le triangle n'est pas rectangle")

rectangle(-2,-2,3,-4,2,8) #pas de besoin de print, celui ci est déjà
#inclu dans la fonction
```

Mise en pratique 4

```
def milieu(xM,yM,xN,yN):
```

```
  xI=(xM+xN)/2
  yI=(yM+yN)/2
  return xI,yI

def parallelogramme(xA,yA,xB,yB,xC,yC,xD,yD):
  if milieu(xA,yA,xC,yC)==milieu(xB,yB,xD,yD):
    print("Le quadrilatère est un parallélogramme.")
  else :
    print("Le quadrilatère n'est pas un parallélogramme.")

parallelogramme(6,-2,9,-1,10,2,7,1) #test avec un parallélogramme
parallelogramme(6,-2,9,-1,10,2,7,-1)#test avec un quadri quelconque

def symetrique(xA,yA,xI,yI):
  xB=2*xI-xA #I étant le milieu de [AB] on a retravaillé les formules
  yB=2*yI-yA #du milieu d'un segment.
  return xB,yB
```

Mise en pratique 5

```
def g(x):
  return x**2-10*x+21

def min(Xmin,Xmax,pas):
  minY=g(Xmin)
  x=Xmin
  xRef=x #première ligne d'un petit bonus non demdandé
  while x<Xmax:
    x=x+pas
    if minY>g(x):
      minY=g(x)
      xRef=x #seconde
  print(f"le minimum de g sur [{Xmin},{Xmax}] est y={minY}")
  print(f"il est atteint en x={xRef}.") #troisième

min(-20,20,0.1) "petit test : recherche du minimum entre -20 et 20."
```

Mise en pratique 6

```
def diviseur(a,m):
  """Entrées : a et m deux entiers
  Sortie : booléen qui indique si a est divisible par m"""
  #ces triples guillemets encadrent la documentation de la fonction
  #c'est facultatif mais ça aidera les gens qui voudront l'utiliser
  if a%m==0: #on regarde si le reste de la division de a
    return True  #par m vaut 0 (a divisible par m) ou pas
  else :
    return False

#version expresse
def diviseuR(a,m):
  return a%m==0 #ce test a naturellement pour résultat True ou False

def simpli(num, deno, k):
  if diviseur(num,k) and diviseur (deno,k):
    print(f"la fraction vaut{num/k}/{deno/k} ")
  else :
    print(f"la fraction vaut{num}/{deno} ")
```

Mise en pratique 7

```
def PGCD(a,b):
  """recherche le pgcd de a et b deux entiers non nuls"""
  dividende = a
  diviseur =b
  reste=1
  while True:
    quotient=dividende//diviseur
    reste=dividende%diviseur
    if reste==0 :
      return diviseur #à ce stade il vaut le dernier reste non nul
    dividende=diviseur
    diviseur = reste
```