جامعة هواري بومدين
للعلوم والتكنولوجيا
U S T H B

Programmation PYTHON

Cours 3

Nassim ZELLAL

2020/2021

Les fonctions/méthodes prédéfinies - type()

- type : cette fonction retourne le type de donnée.

- liste=(4,5,9)
- print(type(liste))
- > <class ‘tuple’>

Les fonctions/méthodes prédéfinies - reverse() / reversed()

- reverse : retourne les éléments d'une liste inversés.
- a =[7,9,8]
- a.reverse()
- print(a)
- > [8,9,7]
- reversed :
- a =[7,9,8]
- print(list(reversed(a)))
- > [8,9,7]

Les fonctions/méthodes prédéfinies - sort()

- sort() : permet de trier une liste.
- texte=["pomme","pain","pomme","orange","pomme","pain"]
- texte.sort()
- print(texte)
- >['orange', 'pain', 'pain', 'pomme', 'pomme', 'pomme']

Les fonctions/méthodes prédéfinies - sorted()

- sorted() : permet de trier une liste ou un dictionnaire.
- texte=["pomme","pain","pomme","orange","pomme","pain"]
- print(sorted(texte))
- >['orange', 'pain', 'pain', 'pomme', 'pomme', 'pomme']
- ----------
- dic={'976':'Mongolia','52':'Mexico','212':'Morocco','64':'New Zealand','33':'France'}
- print(sorted(dic.keys()))#ou bien print(sorted(dic))
- > ['212', '33', '52', '64', '976']

Les fonctions/méthodes prédéfinies - sorted()

- #------Trier les clés----------#
- dic={"cours1":15,"cours2":18,"cours3":11}
- for i in sorted(dic.keys()):#ou bien for i in sorted(dic)
- print(i)
- #------Trier les valeurs---------#
- dic={"cours1":15,"cours2":18,"cours3":11}
- for i in sorted(dic.values()):
- print(i)

Les fonctions/méthodes prédéfinies - sorted()

```
C:\Users\user\Desktop>test.py
cours1
cours2
cours3
11
15
18
```

Les fonctions/méthodes prédéfinies - get()

- dic={"cours1":15,"cours2":18,"cours3":11}
- for i in sorted(dic.keys()):#ou bien for i in sorted(dic)
- print(i,dic.get(i))

- #La méthode get(key) prend comme paramètre la clé et retourne sa valeur.
- #On trie les clés du dictionnaire et pour chaque clé on affiche sa valeur.

Les fonctions/méthodes prédéfinies - get()

```
C:\Users\user\Desktop>test.py
cours1 15
cours2 18
cours3 11

C:\Users\user\Desktop>_
```

Les fonctions/méthodes prédéfinies - get()

- texte=["pomme","pain","pomme","orange","pomme","pain"]
- **lettres ={}**
- for c in texte:
- **if c in lettres:**
- **lettres[c] = lettres[c] + 1#ou lettres[c]+=1**
- **else:**
- **lettres[c] = 1**
- for i in sorted(lettres):
- print(i,lettres.get(i))
- #Ce script remplit le dictionnaire «lettres» avec le contenu de la liste «texte». On utilise une condition pour éviter l'erreur **KeyError:clé**, signifiant que la clé n'existe pas encore.

Les fonctions/méthodes prédéfinies - get()

- texte=["pomme","pain","pomme","orange","pomme","pain"]
- **lettres ={}**
- for c in texte:
- **if c not in lettres:**
- **lettres[c] = 1**
- **else:**
- **lettres[c] = lettres[c] + 1#ou lettres[c]+=1**
- for i in sorted(lettres):
- print(i,lettres.get(i))
- #Ce script remplit le dictionnaire «lettres» avec le contenu de la liste «texte». On utilise une condition pour éviter l'erreur **KeyError:clé**, signifiant que la clé n'existe pas encore.

Les fonctions/méthodes prédéfinies - get()

- texte=["pomme","pain","pomme","orange","pomme","pain"]
- **lettres ={}**
- for c in texte:
- **if c not in lettres:**
- **lettres[c] = 0**
- **lettres[c] +=1**
- for i in sorted(lettres):
- print(i,lettres.get(i))
- #Ce script remplit le dictionnaire «lettres» avec le contenu de la liste «texte». On utilise une condition pour éviter l'erreur **KeyError:clé**, signifiant que la clé n'existe pas encore.

Les fonctions/méthodes prédéfinies - get()

- texte=["pomme","pain","pomme","orange","pomme","pain"]
- lettres ={}
- for c in texte:
- **lettres[c] = lettres.get(c, 0) + 1**
- for i in sorted(lettres):
- print(i,lettres.get(i))
- #Si une clé n'existe pas, get() peut prendre un second argument (ici 0) pour l'afficher dans ce cas. Par défaut, cette valeur est « None » (« None Type »).
- #Ce script remplit le dictionnaire « lettres » avec le contenu de la liste « texte ».

Les fonctions/méthodes prédéfinies - get()

- > orange 1
- pain 3
- pomme 2

Les fonctions/méthodes prédéfinies - items()

- dic={"cours1":15,"cours2":18,"cours3":11}
- print(dic.items())
- >dict_items([('cours1', 15), ('cours2', 18), ('cours3', 11)])

- #Cette méthode prédéfinie retourne les couples (clé,valeur) d'un dictionnaire.

Les fonctions/méthodes prédéfinies – keys() / values()

- dic={"cours1":15,"cours2":18,"cours3":11}
- print (dic.keys()) #affichage des clés d'un dictionnaire
- print (dic.values()) #affichage des valeurs d'un dictionnaire

Les fonctions/méthodes prédéfinies - append()

- append : insère un élément à la fin (à droite) d'une liste.

```
t= [1,2,3,4]
t.append(8)
print(t)
> [1,2,3,4,8]
```

Les fonctions/méthodes prédéfinies - pop()

- pop : supprime un élément à la fin (à droite) d'une liste et le retourne.
- t= [1,2,3,4]
- x=t.pop()
- print(x)
- > 4
- pop peut prendre comme paramètre l'indice de l'élément à supprimer.

t= [1,2,3,4]

x=t.pop(2)

print(x)

> 3

Les fonctions/méthodes prédéfinies - appendleft()

- appendleft : insère un élément au début (à gauche) d'une liste.

```
from collections import deque
t=deque([1,2,3,4]) # création d'une « file à double entrée »
t.appendleft(8)
print(t)
>  deque([8, 1, 2, 3, 4])
#Possibilité d'utiliser append() sur une deque().
```

Les fonctions/méthodes prédéfinies - popleft()

- popleft : supprime un élément au début (à gauche) d'une liste et le retourne.

```
from collections import deque
t=deque([1,2,3,4]) # création d'une « file à double entrée »
x=t.popleft()
print(x)
> 1
#Avec popleft(), pas de possibilité de spécifier un indice comme avec pop(). On peut utiliser pop() mais sans indice.
```

Les fonctions/méthodes prédéfinies - clear()

- clear() : cette méthode permet de vider une liste.

```
st= [123, 'abc', 'efg', 'abc', 123]
st.clear()
print(st)
> [ ]
```

Les fonctions/méthodes prédéfinies - split() / join()

- split : découpe une chaîne de caractères selon un séparateur et met les éléments dans une liste.

```
inf = "Premier:Ministre:Acteur:14, rue Saint Honoré"
a=inf.split(":")
print(a[3])
```

> 14, rue Saint Honoré

- join : renvoie une chaîne de caractères contenant les éléments d'une liste, concaténés et séparés par un délimiteur.

```
sep=","
myNames =["Samir", "Meriem", "Salim"]
print(sep.join(myNames)) # ou bien print(",".join(myNames))
```

- Samir,Meriem,Salim
- Remarque : Il est possible d'intégrer une « expression régulière » dans la méthode split().

Les fonctions/méthodes prédéfinies - len() / index()

- len : renvoie le nombre d'éléments d'une liste.

```
tab=[3,6,9]
 print (len(tab))
> 3
```

- index: détermine la position d'une lettre dans une chaîne de caractères.

```
string = "perlmeme.org"
print(string.index('o'))
```

Si le caractère n'existe pas dans la chaîne, le message «ValueError : substring not found » est affiché.

Les fonctions/méthodes prédéfinies - len() / rstrip()

- len : retourne la longueur en caractères de la valeur d'une variable.

```
tab="Une pomme"
print(len(tab))
> 9
```

- rstrip() : supprime les sauts de ligne.

```
var ="Je suis un étudiant en informatique\n"
print(var.rstrip())
```

Les fonctions/méthodes prédéfnies - upper() / lower()

- upper() : retourne la chaîne en majuscules.

```
a = "table"
print(a.upper())
> TABLE
```

- lower() : retourne la chaîne en minuscules.

```
a = "TABLE"
print(a.lower())
> table
```

Les fonctions/méthodes prédéfinies - count()

- count() : compte le nombre d'occurrences d'une valeur dans une liste.

```
st= [123, 'abc', 'efg', 'abc', 123]
print (st.count('efg'))
print (st.count(123))
> 1
> 2
```

Les fonctions/méthodes prédéfinies - count()

- count() : retourne le nombre d'occurrences d'une sous-chaîne dans une chaîne.

```
ch = "Python est un langage de programmation."
s = "on"
print(ch.count(s))
> 2
```

On peut préciser l'indice de début et l'indice de fin de la recherche.

```
ch = "Python est un langage de programmation."
s = "o"
print(ch.count(s,4,36))
> 3
```

Les fonctions/méthodes prédéfinies - replace()

```
a="Je suis étudiant à l'USTHB"
print(a.replace('USTHB','ESI'))
> Je suis étudiant à l’ESI
```

Création d'une fonction - def

- def f(p1,p2): #en-tête de la fonction
- print(p1+p2) #corps de la fonction

Définition de la fonction

- f(5,6)# appel de la fonction
- > 11
- #----------#
- def date(jour, mois,an):
- return str(jour) +" "+ mois+ " " +str(an)

- print(date(5,"janvier",2000))
- > 5 janvier 2000

Création d'une fonction - def

- t=["5","janvier","2000"]
- def date(tab):
- 	return tab[0] +" "+ tab[1]+ " " +tab[2]
- print(date(t))
- > 5 janvier 2000
- #----------#
- dic={'976':'Mongolia','52':'Mexico','212':'Morocco','64':'New Zealand','33':'France'}
- def date(tab):
- 	return tab.keys()
- print("Résultat ",date(dic))
- > Résultat dict_keys([‘976’, ‘52’,’212’,’64’,’33’])

Création d'une fonction - def

- def date(tab):
- return tab[0] +" "+ tab[1]+ " " +tab[2]
- print(date("5","janvier","2000"))
- > TypeError: date() takes 1 positional argument but 3 were given
- #----------Packing avec *----------#
- def date(*t):
- print(type(t))#affiche <class 'tuple'>
- return str(t[0])+" "+t[1]+ " " +str(t[2])
- print(date(5,"janvier",2000))#empaqueter tous les arguments dans un tuple
- > <class 'tuple'>
- > 5 janvier 2000

Création d'une fonction - def - packing - *

```
def date(*t):
    print(type(t))#affiche <class 'tuple'>
    for i in range(len(t)):
        print(t[i])
date(5,"janvier",2000)
date(10,"juin")
```

Création d'une fonction - def - packing - **

- def date(**tab):
- print(type(tab))
- return tab
- print(date(**prenom**="Sam",**nom**="Gabriel"))
- #empaqueter tous les arguments dans un dictionnaire (dict)
- <class 'dict' >
- {'prenom':'Sam','nom':'Gabriel'}
- #on peut aussi écrire directement sans **** :
- print(date({"prenom":"Sam","nom":"Gabriel"}))

Création d'une fonction - def - unpacking - *

- def f(a,b,c):
- print(a,b,c)
- liste = [1, 2, 3]
- f(liste)
- >TypeError: f() missing 2 required positional arguments : 'b' and 'c'
- #----------#
- def f(a,b,c):
- print(a,b,c)
- liste = [1, 2, 3]
- f(*liste)
- > 1 2 3

Création d'une fonction - def - unpacking - **

- def date(prenom,nom):
- print(prenom,nom)
- d={'prenom':'Sam', 'nom':'Gabriel'}
- date(d)
- > TypeError: date() missing 1 required positional argument : 'nom'
- #----------#
- def date(prenom,nom):
- print(prenom,nom)#deux variable de type « str »
- d={'prenom':'Sam', 'nom':'Gabriel'}
- date(**d)
- date(*d) # avec une seule étoile on affiche les clés
- > Sam Gabriel
- > prenom nom

Création d'une fonction - def - unpacking et packing

- def f(a,b,*c): #packing
- print(a,b,c)
- print(type(c))#affiche <class 'tuple'>
- liste = [1, 2, 3, 4]
- f(*liste)#unpacking
- >1 2 (3, 4)

Création d'une fonction anonyme avec « lambda »

- a="J'ai soif"
- def s(arg):
- print(arg)
- s(a)
- #----------Avec lambda----------#
- s = lambda arg: print("Bonjour "+arg)
- s("j'ai soif")
- > Bonjour j'ai soif
- print((lambda arg: "Bonjour "+arg)("j'ai soif"))
- #On ne peut pas avoir plus d'une instruction dans une fonction lambda.

Création d'une fonction anonyme avec « lambda »

- g = lambda x,y: x*y
- print(g(5,6))
- > 30
- #----------Autre syntaxe----------#
- print((lambda x: x*5)(6))
- > 30
- print((lambda x, y: x + y)(4,6))
- > 10

Tri d'un dictionnaire avec « lambda »

- dic={'976':'Mongolia','52':'Mexico','212':'Morocco','64':'New Zealand','33':'France'}
- dic_trie = sorted(dic.items(), key=lambda x: x[1],reverse=True) #commande « reverse »/expression/mot-clé « lambda »
- for a,b in dic_trie: #Affichage
- print(a,b)
- #Dans ce script, on trie les clés via les valeurs, de Z à A (ordre décroissant/descendant).

Tri d’un dictionnaire avec « lambda »

```
64 New Zealand
212 Morocco
976 Mongolia
52 Mexico
33 France
```

Tri d'un dictionnaire avec « lambda »

- dic={'976':'Mongolia','52':'Mexico','212':'Morocco','64':'New Zealand','33':'France'}
- dic_trie = sorted(dic.items(), key=lambda x: x[0],reverse=True) #commande « reverse »/expression « lambda »
- for a,b in dic_trie: #Affichage
- print(a,b)
- #Dans ce script, on trie les valeurs via les clés, de Z à A (ordre décroissant/descendant).
- Si on met reverse à « True », c'est l'équivalent de :
- for i in reversed(sorted(dic.keys())):#ou bien for i in reversed(sorted(dic))
  - print(i,dic.get(i))
- Si on met reverse à « False », c'est l'équivalent de :
- for i in sorted(dic.keys()):#ou bien for i in sorted(dic)
  - print(i,dic.get(i))

Tri d’un dictionnaire avec « lambda »

Exercice 1

- Réécrire le script suivant pour trier les éléments de la liste « a » par ordre décroissant.

- a = [1,2,3,4,5]
- def f (arg):
- print ("Les chiffres sont ",arg)
- f(a)

Exercice 2

- Modifier le script ci-dessous pour **trier** les clés du dictionnaire « f » par **ordre décroissant**.
- a=["pomme","pain","pomme","orange","pomme","pain"]
- f={}
- for c in a:
- 	f[c] = f.get(c, 0) + 1
- for i in sorted(f):#ou bien for i in sorted(f.keys())
- 	print(i,f.get(i))

Exercice 3

- **Faire un script Python**
- **Entrer en paramètre/argument:**
- **défragmentation, reduplication et colocation.**
- **À faire :**
- **Compter le nb de caractères de chaque mot.**
- **Séparer les affixes.**
- **Compter le nb de caractères de chaque racine.**
- **Afficher les mots de la façon suivante :**
- **préfixe+racine+suffixe nbre caractères mot, nbre caractères racine**

Exercice 4

- Écrire un script générant la table de multiplication (de 1 à 10) d'un nombre entier passé en argument à une fonction.

Exercice 5

- Écrire un script qui prend l'argument « 56cd ».
- Afficher sur votre console/invite de commandes :
  - Un caractère 5
  - Un caractère 6
  - Un caractère c
  - Un caractère d

Mon courriel

zellal.nassim@gmail.com