CONDITIONS DU JOUR-9

Conditionnels

Par défaut, les instructions du script Python sont exécutées séquentiellement de haut en bas. S i la logique de traitement le nécessite, le flux d'exécution séquentiel peut être modifié en deux manières:

•Exécution conditionnelle: un bloc d'une ou plusieurs instructions sera exécuté si une certa ine expression est vraie • Exécution répétitive: un bloc d'une ou plusieurs instructions sera exécuté de manière répétitive tant qu'une certaine expression est vraie. Dans cette section, nous couvrirons les instructions *if*, *else*, *elif*. La comparaison et les opérateurs logiques qu e nous avons appris dans les sections précédentes seront utiles ici.

Si condition

Dans Python et d'autres langages de programmation, le mot clé *if* est utilisé pour vérifier si un e condition est vraie et pour exécuter le code de bloc. Rappelez-vous l'indentation après le côl on.

```
# Syntaxe Si Condition: cette partie du code s'exécute pour des conditions
de vérité
```

Exemple: 1

```
a = 3 Si a > 0: print ('a est un nombre positif')


# A est un nombre positif
```

Comme vous pouvez le voir dans l'exemple ci-dessus, 3 est supérieur à 0. La condition était vraie et le code de bloc a été exécuté. Cependant, si la condition est fausse, nous ne voyons pas le résultat. Afin de voir le résultat de la condition de fausseté, nous devrions avoir un aut re bloc, qui va être *else*.

Si bien

Si la condition est vraie, le premier bloc sera exécuté, sinon la condition Else s'exécutera.

```
# Syntaxe Si Condition: Cette partie du code s'exécute pour des condition
s de vérité d'autre: cette partie du code s'exécute pour de fausses condition
s
```

Exemple:

```
a = 3 Si a < 0: print ('a est un nombre négatif')


else: print ('a est un nombre positif')
```

La condition ci-dessus s'avère fausse, le bloc ELSE a donc été exécuté. Que diriez-vous si notr e état est supérieur à deux? Nous pourrions utiliser ***elif***.

Si Elif d'autre

Dans notre vie quotidienne, nous prenons des décisions au quotidien. Nous prenons des décisio ns non pas en vérifiant une ou deux conditions mais plusieurs conditions. Comme similaire à la vie, la programmation est également pleine de conditions. Nous utilisons ***elif*** lorsque nous avo ns plusieurs conditions.

```
# syntax
if condition:
    code
elif condition:
    code
else:
    code
```

Exemple:

```
a = 0 si a > 0: print ('a est un nombre positif')


elif a < 0: print ('a est un nombre négatif')

else: print ('a is zéro')
```

Main courte

```
# code de syntaxe si condition autre code
```

Exemple:

```
a = 3 print ('a est positif') si a > 0 else print ('a is négatif') # la première condition met, 'a est p ositif' sera imprimé
```

Conditions imbriquées

Les conditions peuvent être imbriquées

```
# Syntaxe si condition: cod
e si condition: code
```

Exemple:

```
a = 0 Si a > 0: si a% 2 == 0: print ('a est un positif et même entier') else: pri
nt ('a est un nombre positif') elif a == 0: print ('a is zero') else: print ('a est u
n nombre négatif')
```

Nous pouvons éviter d'écrire une condition imbriquée en utilisant l'opérateur logique *and*.

Si les opérateurs de condition et logique

```
# Syntaxe Si condition et condition: code
```

Exemple:

```
a = 0 Si a > 0 et un% 2 == 0: print ('a est un entier uniforme et positif') elif a
> 0 et a% 2! = 0: print ('a est un entier positif') elif a == 0: print ('a is zero') e
lse: print ('a est négatif')
```

Si et ou des opérateurs logiques

```
# Syntaxe si condition ou condition: cod
e
```

Exemple:

```
Utilisateur = 'James' Acc
ess_Level = 3
```

```
Si l'utilisateur == 'admin' ou access_level >= 4:
Imprimer («Accès accordé!») Else: Print («Accès
refusé!»)
```

Vous faites très bien, n'abandonnez jamais parce que les grandes choses prennent du temps. Vous venez de terminer les défis du jour 9 et vous êtes 9 étapes en route vers votre chemin ver s la grandeur. Faites maintenant quelques exercices pour votre cerveau et vos muscles.

Exercices: Jour 9

Exercices: niveau 1

1. Obtenez la saisie de l'utilisateur à l'aide de l'entrée («Entrez votre âge:»). Si l'utilisate ur a 18 ans ou plus, donnez des commentaires: vous êtes assez vieux pour conduire. Si moins de 18, donnez des commentaires pour attendre le nombre manquant d'années. Sor tir:

```
Entrez votre âge: 30 Vous êtes assez vieux pour apprendre à
conduire.  Sortie: Entrez votre âge: 15 Vous avez besoin de
3 ans de plus pour apprendre à conduire.
```

2. Comparez les valeurs de My_age et de votre_age en utilisant si... sinon. Wh est plus â gé (moi ou vous)? Utilisez l'entrée («Entrez votre âge:») pour obtenir l'âge en entrée. Vo us pouvez utiliser une condition imbriquée pour imprimer une «année» pour une différe nce d'âge, des «années» pour des différences plus importantes et un texte personnalisé si My_age = votre_age. Sortir:

```
Entrez votre âge: 30 Vous avez 5 ans de plus q
ue moi.
```

3. Obtenez deux numéros de l'utilisateur à l'aide de l'invite d'entrée. Si A est supérieur à B, le retour a est supérieur à B, si A est moins B, le retour a est plus petit que b, sin on a est égal à b. Sortir:

```
Entrez numéro un: 4 Entrez n
uméro deux: 3 4 est supérieur
à 3
```

Exercices: niveau 2

1. Écrivez un code qui donne des notes aux élèves en fonction de leurs scores:

```
80-100, A 70-8
9, B 60-69, C
50-59, D 0-49,
F
```

2. Vérifiez si la saison est automne, w inter, printemps ou été. Si l'entrée de l'utilisateur est : septembre, octobre ou novembre, la saison est automnale. Décembre, janvier ou février, l a saison est w inter. Mars, avril ou mai, la saison est au printemps juin, juillet ou août, la sa ison est de l'été

3. La liste suivante contient des fruits:

```sh
fruits = ['banana', 'orange', 'mango', 'lemon']
```

Si un fruit n'existe pas dans la liste, ajoutez le fruit à la liste et imprimez la liste modifiée. Si le fruit existe imprimé («ce fruit existe déjà dans la liste»)

Exercices: niveau 3

1. Ici, nous avons un dictionnaire de personne. N'hésitez pas à le modifier!

```
personne = {'premier_name': 'asabeneh', 'last_name': 'yetayeh', 'age': 250, 'country': 'Fi
nland', 'is_marred': true, 'compétences': 'javascript' 'Zipcode': '02210'}}
```

* Vérifiez si le dictionnaire de la personne a des compétences, si c'est le cas, imprimez les co
mpétences intermédiaires dans la liste des compétences.
* Vérifiez si le dictionnaire de la personne a une clé de compétences, si c'est le cas, vérifiez si la personne a des compétences «Python» et imprimez le résultat.
* Si A Person Skills n'a que JavaScript et React, Print («Il est un développeur frontal»), si les c
ompétences de la personne ont Node, Python, MongoDB, Print (`` Il est un développeur backen
d "), si la personne Skills a React, Node et MongoDB, Print (`` Il est un développeur Fullstack '
), Else Print (' INCONNET TITME ') - Pour plus de résultats plus précis! * Si la personne est mariée et si elle vit en Finlande, imprimez

Les informations dans le format suivant:

    Asabeneh Yetayeh vit en Finlande. Il est marié.

🥕 CONGRATULATIONS ! 🥕

Boucles du jour-10

Boucles

La vie est pleine de routines. En programmation, nous effectuons également beaucoup de tâc hes répétitives. Afin de gérer les langages de programmation de tâches répétitifs, utilisez des boucles. Le langage de programmation Python fournit également les types de deux boucles s uivants:

1. Pendant la boucle
2. Pour Loop

Pendant la boucle

Nous utilisons le mot réservé *while* pour faire une boucle de temps. Il est utilisé pour exécut er un bloc d'instructions à plusieurs reprises jusqu'à ce qu'une condition donnée soit satisfait e. Lorsque la condition devient fausse, les lignes de code après la fin de la boucle se poursui vront.

```
# Syntaxe pendant la conditi
on: le code va ici
```

Exemple:

```
count = 0 tandis que le nombre <
5: print (count) count = count +
1 #prints de 0 à 4
```

Dans ce qui précède, la boucle, la condition devient fausse lorsque le nombre est de 5. C'est à ce moment que la boucle s'arrête. Si nous sommes intéressés à exécuter le bloc de code une fois que la condition n'est plus vraie, nous pouvons utiliser *else*.

```
  # syntax
while condition:
    code goes here
else:
    code goes here
```

Exemple:

```
Count = 0 tandis que le nombre
< 5: print (count) count = count
+ 1 else: print (count)
```

La condition de boucle ci-dessus sera fausse lorsque le nombre est 5 et la boucle s'arrêt e, et l'exécution démarre l'instruction ELSE. En conséquence, 5 sera imprimé.

Break and Contin - Partie 1

- Break: Nous utilisons la pause lorsque nous aimons sortir de la boucle ou arrêter la boucle.

```
# Syntaxe pendant la condition: le co
de va ici si un autre_condition:



casser
```

Exemple:

```
Count = 0 tandis que le nombre
< 5: print (count) compter = cou
nt + 1 if count == 3: pause
```

La boucle ci-dessus est uniquement imprimée 0, 1, 2, mais quand elle atteint 3, elle s'arrête.

- Continuez: avec l'instruction Continuer, nous pouvons ignorer l'itération actuelle et cont inuer avec la suivante:

```
# Syntaxe pendant la condition: le co
de va ici si un autre_condition:



continuer
```

Exemple:

```
Count = 0 tandis que le nombre < 5: If
Count == 3: Count = Count + 1 Contin
uer Print (Count) Count = Count + 1
```

La boucle ci-dessus imprime uniquement 0, 1, 2 et 4 (sauter 3).

Pour boucle

Un mot-clé *for* est utilisé pour faire une boucle pour une boucle, similaire avec d'autres langag es de programmation, mais avec certaines différences de syntaxe. La boucle est utilisée pour it ération sur une séquence (c'est soit une liste, un tuple, un dictionnaire, un ensemble ou une cha îne).

- Pour boucle avec liste

```
# Syntaxe pour itérateur en L
ST:
Le code va ici
```

Exemple:

```
Numéros = [0, 1, 2, 3, 4, 5] Pour le nombre en numéros: # Le numéro est un nom temporaire pour se référer aux éléments de la liste, valide uniquement à l'intérieur de cette boucle imprim er (numéro) # Les numéros seront imprimés en ligne par ligne, de 0 à 5
```

- Pour boucle avec chaîne

```
# syntaxe
Pour Iterator dans String: le code va
ici
```

Exemple:

```
Langue = 'Python' pour la lettre en l
angue: imprimer (lettre)



pour I à portée (Len (langue)):
imprimer (langue [i])
```

- Pour boucle avec tuple

syntaxe

```
for iterator in tpl:
    code goes here
```

Exemple:

```
Nombres = (0, 1, 2, 3, 4, 5) pour le nombre
en numéros: imprimer (numéro)
```

- Pour une boucle avec un dictionnaire en boucle à travers un dictionnaire vous donne la cl é du dictionnaire.

```
# Syntaxe pour Iterator dans D
CT: le code va ici
```

Exemple:

```
personne = {'first_name': 'asabeneh',

'last_name': 'encoreayeh', 'Âge': 250,
```

'Country': 'Finlande', 'is_marred': true, 'compétences': ['javascript', 'react', 'node', 'mongodb', ' python'], 'adresse': {street ':' Street ',' zipcode ':' 02210 '}} pour la clé: la clé: la clé) pour la c lé, la valeur, la valeur en personne. Nous obtenons les touches et les valeurs imprimées • bo ucles dans Set # Syntaxe

Pour Iterator dans ST: Code v
a ici

Exemple:

IT_COMPANIES = {'Facebook', 'Google', 'Microsoft', 'Apple', 'IBM', 'Oracle', 'Amazon'}
pour la société dans IT_COMPANIES: print (Company)

Break and Contin - Partie 2

Rappel court: *Break*: Nous utilisons Break lorsque nous aimons arrêter notre boucle av ant sa fin.

Syntaxe pour itérateur en séquence:

Le code va ici si la condition
: pause

Exemple:

Nombres = (0,1,2,3,4,5) pour le no
mbre en nombres: imprimer (numér
o) si numéro == 3: pause

Dans l'exemple ci-dessus, la boucle s'arrête lorsqu'elle atteint 3.

Continuez: nous utilisons Continuer lorsque nous aimons ignorer certaines des étapes de l'itér ation de la boucle.

```
# Syntaxe pour itérateur en séquence: l
e code va ici si condition: Continue
```

Exemple:

```
Numéros = (0,1,2,3,4,5) pour le nombre en nombres: imprimer (numéro) si numéro == 3: Cont inuez Print ('Le numéro suivant devrait être', le numéro + 1) si le numéro! = 5 Else Print ("Loo p's End") # pour les conditions courtes de la main nécessite les deux si et bien les statuts impri ment ('Extérieur de la boucle'))
```

Dans l'exemple ci-dessus, si le nombre est égal à 3, l'étape *after* La condition (mais à l'intérieur de la boucle) est ignorée et l'exécution de la boucle continue s'il reste des itérations.

La fonction de plage

La fonction *range()* est utilisée sur la liste des nombres. Le *range(start, end, step)* prend tr ois paramètres: Démarrage, fin et incrément. Par défaut, il part à partir de 0 et l'incrément est 1. La séquence de plage nécessite au moins 1 argument (fin). Création de séquences à l'aide d e la plage

```
LIST = (plage (11)) imprimer (LST) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] ST = set (Range (1, 11)) # 2 Arguments indiquent le démarrage et la fin de la séquence, pas sur défaut 1 print (ST) # {1, 3 , 4, 5, 6, 7, 8, 9, 10} LST = list (plage (0,11,2)) print (lst) # [0, 2, 4, 6, 8, 10] st = set (plage (0,1 1,2)) print (st) # {0, 2, 4, 6, 8, 10} # syntaxe pour iterator in plage (start, end, step):
```

Exemple:

```
pour le nombre dans la gamme (11): imprimer (numéro) # imprime 0 à 10, sans co
mpter 11
```

Imbriqué pour la boucle

Nous pouvons écrire des boucles dans une boucle.

```
# Syntaxe pour x en y: p
our t dans x: imprimer (
t)
```

Exemple:

```
personne = {'premier_name': 'asabeneh', 'last_name': 'yetayeh', 'age': 250, 'country': 'Fi
nland', 'is_marred': true, 'compétences': 'javascript' 'Zipcode': '02210'}} pour la clé en
personne: si clé == 'compétences': pour la compétence en personne ['compétences']: i
mprimer (compétence)
```

Pour ailleurs

Si nous voulons exécuter un message à la fin de la boucle, nous utilisons d'autre.

```
# Syntaxe pour iterator dans la gamme (démarrage, fin, étape)
: faire autre chose: imprimer («la boucle terminée»)
```

Exemple:

```
pour le nombre dans la gamme (11): imprimer (numéro) # imprime 0 à 10, sans c
ompter 11
else: imprimer («la boucle s'arrête à», numéro)
```

Passer

Dans Python, lorsque l'instruction est requise (après le demi-colon), mais nous n'aimons pas exécuter de code là-bas, nous pouvons écrire le mot `pass` pour éviter les erreurs. Nous pouv ons également l'utiliser comme espace réservé, pour les futures déclarations.

Exemple:

```
pour le nombre dans la plage (6): pa
sser
```

Vous avez établi une grande étape, vous êtes imparable. Continue! Vous venez de terminer l es défis du jour 10 et vous êtes à 10 pas de la grandeur à la grandeur. Faites maintenant quelqu es exercices pour votre cerveau et vos muscles.

Exercices: Jour 10

Exercices: niveau 1

1. Itérer 0 à 10 en utilisant pour la boucle, faites de même en utilisant la boucle.

2. Iléter 10 à 0 en utilisant pour la boucle, faites de même en utilisant la boucle pendant.

3. Écrivez une boucle qui fait sept appels à imprimer (), nous obtenons donc sur la sor tie le triangle suivant:

```
#
##
###
####
#####
######
#######
```

4. Utilisez des boucles imbriquées pour créer ce qui suit:

```
# # # # # # # # # # # # #
# # # # # # # # # # # # #
# # # # # # # # # # # # #
# # # # # # # # # # # # #
# # # # # # # # # # # # #
#
```

5. Imprimez le modèle suivant:

```
0 x 0 = 0 1 x 1 = 1 2
x 2 = 4 3 x 3 = 9 4 x
4 = 16 5 x 5 = 25 6 x
6 = 36 7 x 7 = 49 8 x
8 = 64 9 x 9 = 81 10
x 10 =} 64 9 X 9 = 8
1 10 x 10 =
```

6. itérer dans la liste, [«python», «numpy», «pandas», «django», «flask»] en utilisant une b oucle pour une boucle et imprimez les éléments. 7. Utiliser pour la boucle pour itérer de 0 à 100 et imprimer uniquement les nombres 8. Utiliser pour la boucle pour itérer de 0 à 100 et imprimer uniquement les nombres impairs

Exercices: niveau 2

1. Utiliser pour la boucle pour itérer de 0 à 100 et imprimer la somme de tous les nombres.

La somme de tous les nombres est 5050.

1. Utilisez pour la boucle pour itérer de 0 à 100 et imprimez la somme de tous les even s et la somme de toutes les chances.

La somme de tous les evens est de 2550. Et la somme de toutes les chances est de 2500.

Exercices: niveau 3

1. Accédez au dossier de données et utilisez le fichier Pays.py. Boucle à travers les pays et extraire tous les pays contenant le mot *land*. 2. Il s'agit d'une liste de fruits, [«banane», «or ange», «mangue», «citron»] inversent l'ordre à l'aide de la boucle. 3. Accédez au dossier d e données et utilisez le fichier Pays_data.py.

---

je. Quel est le nombre total de langues dans les données II. T rouvez les dix langues les plus parlées des données III. Trouve z les 10 pays les plus peuplés du monde

Félicitations!

Fonctions du jour-11

Fonctions

Jusqu'à présent, nous avons vu de nombreuses fonctions Python intégrées. Dans cette section, nous nous concentrerons sur les fonctions personnalisées. Quelle est une fonction? Avant de c ommencer à faire des fonctions, apprenons ce qu'est une fonction et pourquoi nous en avons b esoin?

Définir une fonction

Une fonction est un bloc réutilisable d'instructions de code ou de programmation conçues pour effectuer une certaine tâche. Pour définir ou déclarer une fonction, Python fournit le mot-clé *def*. Ce qui suit est la syntaxe pour définir une fonction. Le bloc fonction du cod e est exécuté uniquement si la fonction est appelée ou invoquée.

Déclarer et appeler une fonction

Lorsque nous faisons une fonction, nous l'appelons déclarant une fonction. Lorsque nous com mençons à utiliser l'IT, nous l'appelons *calling* ou *invoking* une fonction. La fonction peut êtr e déclarée avec ou sans paramètres.

```
# syntaxe # déclarant une fonction
def function_name (): codes codes
# appelant une fonction function_n
ame ()
```

Fonction sans paramètres

La fonction peut être déclarée sans paramètres.

Exemple:

```
Def generate_full_name (): premier_name = 'asabeneh' last_name = 'ye
tayeh' espace = '' full_name = first_name + Space + last_name print (f
ull_name) Generate_full_name () # appelle un fonction ajout ajout_tw
o): Generate_full_name () # appelle a fonction afaf ad_two num_one =
2 num_two = 3 total = num_one + num_two print (total) add_two_nu
mbers ()
```

Fonction Renvoi d'une valeur - Partie 1

La fonction peut également renvoyer des valeurs, si une fonction n'a pas d'instruction de reto ur, la valeur de la fonction n'est aucune. Laissez-nous réécrire les fonctions ci-dessus à l'aide de retour. À partir de maintenant, nous obtenons une valeur à partir d'une fonction lorsque n ous appelons la fonction et l'imprimons.

```
Def Generate_full_name (): premier_name = 'asabeneh' last_name = 'e
ncoreayeh' espace = " full_name = first_name + Space + last_name ret
urn full_name print (generate_full_name ()) def ajout = 2 num_two = 3
total = num_one + num_two return total print (add_two_numbers ())
```

Fonction avec paramètres

Dans une fonction, nous pouvons transmettre différents types de données (numéro, chaîne, booléen, liste, tuple, dictionnaire ou ensemble) en tant que paramètre

- Paramètre unique: si notre fonction prend un paramètre, nous devons appeler notr e fonction avec un argument

```
# syntaxe
# Déclarant une fonction
Def function_name (paramètre):
codes
codes
# Fonction d'appel
print (function_name (argument))
```

Exemple:

```
Def Greetings (nom):
```

```
Message = name + ', bienvenue à Python pour tout le monde!'      return message print
(salutations ('asabeneh')) def add_ten (num): dix = 10 return num + ten print (add_ten
(90)) def square_number (x): return x * x print (square_number (2))
```

```
DEF Area_Of_Circle (R): PI = 3.14
Zone = Pi * R ** 2 RETOUR ZON
E

print (Area_Of_Circle (10))

def sum_of_numbers (n): total = 0 po
ur i dans la plage (n + 1): total += i i
mprimer (total)


print (sum_of_numbers (10)) # 55 PRINT (SUM_
OF_NUMBERS (100)) # 5050
```

- Deux paramètres: une fonction peut ou non avoir un paramètre ou des paramètres. Une fo nction peut également avoir deux paramètres ou plus. Si notre fonction prend des paramètr es, nous devons l'appeler avec des arguments. Vérifions une fonction avec deux paramètre s:

```
# syntaxe
# Déclarant une fonction
Def function_name (para1, para2):
codes
codes
# Fonction d'appel
print (function_name (arg1, arg2))
```

Exemple:

```
Def Generate_full_name (first_name, last_name): espace = " full_name = first_name + espace
+ last_name return full_name print ('full name:', generate_full_name ('asabeneh', 'yeayeh')))
```

```
def sum_two_numbers (num_one, num_two):
sum = num_one + num_two return sum
```

```
print ('somme de deux nombres:', sum_two_numbers (1, 9))

Def Calculate_age (Current_year, Birth_year): Age = Current_year -
Birth_year Return Age;

print ('Âge:', Calculate_age (2021, 1819)))


def poids_of_object (masse, gravité): poids = str (masse * gravity) + 'n' # La valeur doit être
changée en une chaîne de retour en poids de retour ('poids d'un objet dans les newtons:', poi
ds_of_object (100, 9.81)))
```

Passer des arguments avec clé et valeur

Si nous passons les arguments avec clé et valeur, l'ordre des arguments n'a pas d'importance
.

```
# syntaxe # déclarant une fonction def function_name (para1, para2): codes codes # app
elle function print (function_name (para1 = 'John', para2 = 'doe')) # L'ordre des argume
nts n'a pas d'importance ici
```

Exemple:

```
def print_fullName (FirstName, LastName): Space = " full_name = FirstName + Space +
LastName PRINT (Full_name) Print (print_fullname (FirstName = 'Asabeneh', LastName
= 'Yelayeh')) Def Add_two_nUM num2): total = num1 + num2 print (total) print (add_tw
o_numbers (num2 = 3, num1 = 2)) # L'ordre n'a pas d'importance
```

Fonction renvoyant une valeur - partie 2

Si nous ne renvoyons pas de valeur avec une fonction, notre fonction renvoie *None* par défa ut. Pour renvoyer une valeur avec une fonction, nous utilisons le mot-clé *return* suivi de la v ariable que nous renvoyons. Nous pouvons retourner tout type de types de données à partir d' une fonction.

- Renvoi d'une chaîne: Exemple:

```
def print_name (FirstName): retourne premier nom
print_name ('asabeneh') # asabeneh



def print_full_name (FirstName, LastName): Space = " full_name =
FirstName + Space + LastName return full_name



print_full_name (FirstName = 'Asabeneh', LastName = 'encoreayeh')
```

- Renvoi d'un nombre:

Exemple:

```
def add_two_numbers (num1, num2): total = num1
+ num2 return total print (add_two_numbers (2, 3))



Def Calculate_age (Current_year, Birth_year): Age = current_year -
naissance_year Return Age;

print ('Âge:', Calculate_age (2019, 1819)))
```

- Rendre un booléen: Exemple:

```
def is_even (n): si n% 2 == 0: print ('même') return true # return stops davantage l'exécution de
la fonction, similaire à briser return false print (is_even (10)) # true print (is_even (7)) # false
```

- Renvoi d'une liste: Exemple:

```
def find_even_numbers (n): evens = [] po
ur i dans la plage (n + 1): si i% 2 == 0: ev
ens.append (i) return evens



print (find_even_numbers (10))
```

Fonction avec paramètres par défaut

Parfois, nous passons des valeurs par défaut aux paramètres, lorsque nous invoquons la fon ction. Si nous ne transmettant pas les arguments lors de l'appel de la fonction, leurs valeurs par défaut seront utilisées.

```
# syntaxe # déclarant une fonction def function_na
me (param = valeur): codes codes # appelle functio
n function_name () function_name (arg)
```

Exemple:

```
Def Greetings (Name = 'Peter'): Message = name + ', bienvenue à Python pour tout le
monde!'      Return Message Print (salutations ()) imprimer (salutations ('asabeneh')))
```

```
Def Generate_full_name (first_name = 'asabeneh', last_name = 'yetayeh'): espace = " full_n
ame = first_name + Space + last_name return full_name}
```

```
print (generate_full_name ()) imprimer (generate_full_name ('d
avid', 'smith'))

Def Calculate_age (Birth_year, current_year = 2021): Age = current_year - Bi
rth_year Return Age;

Imprimer ('Age:', Calculate_age (1821)) def poids_of_object (masse, gravité = 9.81): poids
= str (masse * gravité) + 'n' # La valeur doit être modifiée en première chaîne de retour de p
oids imprimé ('poids d'un objet dans la terre:', weight_of_object Imprimer ('poids d'un objet
dans les newtons:', poids_of_object (100, 1,62)) # Gravité à la surface de la lune
```

Nombre arbitraire d'arguments

Si nous ne connaissons pas le nombre d'arguments que nous transmettons à notre fonction, nou s pouvons créer une fonction qui peut prendre un nombre arbitraire d'arguments en ajoutant * a vant le nom du paramètre.

```
# syntaxe # déclarant une fonction def function_name (* args)
: codes codes # appelle function function_name (param1, para
m2, param3, ..)
```

Exemple:

```
def sum_all_nums (* nums): total = 0 pour num in num: total += num # identique à
Total = total + num return total print (sum_all_nums (2, 3, 5)) # 10
```

Nombre par défaut et arbitraire de paramètres dans les fonctions

```
Def Generate_Groups (Team, * Args): Print (Team) pour I
dans Args: Print (i) Print (Generate_Groups ('Team- 1', 'As
abeneh', 'Brook', 'David', 'EYOB')))
```

Fonction comme un paramètre d'une autre fonction

```
#Vous pouvez passer les fonctions comme paramètres def carré_nu
mber (n): retour n * n def do_something (f, x): return f (x) imprimer
(do_something (carré_number, 3)) # 27
```

Vous avez réalisé beaucoup jusqu'à présent. Continue! Vous venez de terminer les défis d u jour 11 et vous êtes à 11 étapes de la grandeur à la grandeur. Faites maintenant quelques e xercices pour votre cerveau et vos muscles.

Témoignage

Il est maintenant temps d'exprimer vos réflexions sur l'auteur et le 30 jours. Vous pouvez lais ser votre témoignage sur ce lien

Exercices: Jour 11

Exercices: niveau 1

1. Déclarer une fonction *add_two_numbers*. Il prend deux paramètres et il renvoie une so mme.

2. La zone d'un cercle est calculée comme suit: zone = π x r x r. Écrivez une fonction qui c alcule *area_of_circle*. 3. Écrivez une fonction appelée add_all_nums qui prend un nombr e arbitraire d'arguments et résume tous les arguments. Vérifiez si tous les éléments de la lis te sont des types de nombres. Sinon, donnez un commentaire raisonnable. 4. La températu re en ° C peut être convertie en ° F en utilisant cette formule: ° F = (° C X 9/5) + 32. Écriv ez une fonction qui convertit ° C en ° F, *convert_celsius_to-fahrenheit*. 5. Écrivez une f onction appelée Check-Season, il faut un paramètre de mois et renvoie la saison: automne, w inter, printemps ou été. 6. Écrivez une fonction appelée calcul_slope qui renvoie la pent e d'une équation linéaire 7. L'équation quadratique est calculée comme suit: $ax^2 + bx + c = 0$. Écrivez une fonction qui calcule le jeu de solutions d'une équation quadratique, *solve_quadratic_eqn*. 8. Déclarer une fonction nommée print_list. Il prend une liste en t ant que paramètre et il imprime chaque élément de la liste. 9. Déclarer une fonction nomm ée Reverse_List. Il prend un tableau en tant que paramètre et il renvoie l'inverse du tableau (utilisez des boucles).

```
print (reverse_list ([1, 2, 3, 4, 5])) # [5, 4, 3, 2, 1] impri
mer (reverse_list1 (["a", "b", "c"]))

# ["C", "B", "A"]
```

10. Déclarer une fonction nommée CAPITalize_list_items. Il prend une liste en tant que paramètre et il renvoie une liste majuscule des éléments

11. Déclarer une fonction nommée add_item. Il faut une liste et un paramètre d'élément. I l renvoie une liste avec l'élément ajouté à la fin.

```
Food_staff = ['Potato', 'Tomato', 'Mango', 'Milk'] imprimer (add_item (aliments_staff, 'viande'))
# ['Potato', 'Tomato', 'mango', 'lait', 'viande'] nombres = [2, 3, 7, 9] print (add_item (nombre, 5))
[2, 3, 7, 5, 5, 5]
```

12. Déclarer une fonction nommée Remove_Item. Il faut une liste et un para mètre d'élément. Il renvoie une liste avec l'élément supprimé.

```
Food_staff = ['Potato', 'Tomato', 'Mango', 'Milk'] imprimer (retire_item (aliment
s_staff, 'mango')) # ['Potato',
«Tomate», «lait»];  Nombres = [2, 3, 7, 9] Imprimer (supprimer_it
em (nombres, 3)) # [2, 7, 9]
```

13. Déclarer une fonction nommée SUM_OF_NUMBERS. Il prend un paramètre de nombr e et il ajoute tous les nombres de cette plage.

```
print (sum_of_numbers (5)) # 15 PRINT (SUM_O
F_NUMBERS (10)) # 55 PRINT (SUM_OF_NUM
BERS (100)) # 5050
```

14. Déclarer une fonction nommée SUM_OF_ODDS. Il prend un paramètre de nombre e t ajoute tous les nombres impairs de cette plage.

15. Déclarer une fonction nommée SUM_OF_EVEN. Il prend un paramètre de nombre e t il ajoute tous les nombres pair dans cette plage.

Exercices: niveau 2

1. Déclarez une fonction nommée evens_and_odds. Il prend un entier positif comme p aramètre et il compte le nombre d'evens et les cotes dans le nombre.

```
 print (evens_and_odds (100))
 # Le nombre de cotes est de 50.
 # Le nombre d'Evens est de 51.
```

1. Appelez votre fonction factorielle, il prend un numéro entier en tant que paramètre et il renvoie un factoriel du numéro

2. Appelez votre fonction *is_empty*, il prend un paramètre et il vérifie s'il est vide ou non

3. Écrivez différentes fonctions qui prennent des listes. Ils doivent calculer_mean, ca lculer_median, calculer_mode, calculer_range, calculer_variance, calculer_std (écart -type).

Exercices: niveau 3

1. Écrivez une fonction appelée IS_PRIME, qui vérifie si un nombre est premier.

2. Écrivez une fonction qui vérifie si tous les éléments sont uniques dans la liste.

3. Écrivez une fonction qui vérifie si tous les éléments de la liste sont du même type de do nnées. 4. Écrivez une fonction qui vérifie si la variable fournie est une variable Python va lide 5. Accédez au dossier de données et accédez au fichier Pays-data.py. • Créez une fon ction appelée la plus_pose au monde. Il doit renvoyer 10 ou 20 langues les plus parlées du monde dans l'ordre descendant • créer une fonction appelée la plus_populaled_countries. I l devrait retourner 10 ou 20 pays les plus peuplés dans l'ordre descendant.

Félicitations!