
Révisions python - écrire ses programmes (1)

BARBIER J.M. - LGT Dumezil - TNSI

1 Tests et fonctions

1.1 Exercice 1

Écrire une fonction `est_pair(n)` qui renvoie `True` si l'entier `n` est pair, `False` sinon.

```
est_pair(4)  # renvoie True
est_pair(7)  # renvoie False

# Votre code

# Tests
assert est_pair(4) == True
assert est_pair(7) == False
assert est_pair(0) == True
assert est_pair(-3) == False
print("Tests OK")
```

1.2 Exercice 2

Écrire une fonction `mention(note)` qui renvoie "TB" (16 et plus), "B" (14 et plus), "AB" (12 et plus), "passable" (10 et plus) ou "recalé".

```
mention(15)  # renvoie 'B'
mention(9.5) # renvoie 'recalé'

# Votre code

# Tests
assert mention(18) == "TB"
assert mention(16) == "TB"
assert mention(15) == "B"
assert mention(12) == "AB"
assert mention(10) == "passable"
assert mention(9.5) == "recalé"
print("Tests OK")
```

1.3 Exercice 3

Écrire une fonction `prix_billet(age, etudiant=False)` qui renvoie le prix d'un billet de cinéma :

- moins de 12 ans : 5 €
- étudiant, ou 65 ans et plus : 8 €
- sinon : 12 €

```
prix_billet(30)          # renvoie 12
prix_billet(20, True)     # renvoie 8
prix_billet(etudiant=True, age=10) # renvoie 5

# Votre code
```

```
# Tests
assert prix_billet(30) == 12
assert prix_billet(20, True) == 8
assert prix_billet(70) == 8
assert prix_billet(8) == 5
assert prix_billet(etudiant=True, age=10) == 5
print("Tests OK")
```

2 Boucles while

2.1 Exercice 4

Écrire une fonction `somme_jusqu_a(n)` qui renvoie $1 + 2 + \dots + n$, avec une boucle `while`.

```
somme_jusqu_a(4) # renvoie 10

# Votre code

# Tests
assert somme_jusqu_a(4) == 10
assert somme_jusqu_a(1) == 1
assert somme_jusqu_a(0) == 0
assert somme_jusqu_a(100) == 5050
print("Tests OK")
```

2.2 Exercice 5

Écrire une fonction `nb_chiffres(n)` qui renvoie le nombre de chiffres de l'entier positif n , sans le convertir en chaîne.

Indice : `n // 10` enlève le dernier chiffre.

```
nb_chiffres(2026) # renvoie 4
nb_chiffres(7)   # renvoie 1

# Votre code

# Tests
assert nb_chiffres(2026) == 4
assert nb_chiffres(7) == 1
assert nb_chiffres(0) == 1
assert nb_chiffres(10) == 2
assert nb_chiffres(99999) == 5
print("Tests OK")
```

2.3 Exercice 6

Écrire une fonction `premier_depassement(t, seuil)` qui renvoie l'**indice** du premier élément de `t` strictement supérieur à `seuil`, ou `None` s'il n'y en a pas. Utiliser une boucle `while`, qui s'arrête dès qu'on a trouvé.

```
premier_depassement([3, 8, 12, 20], 10) # renvoie 2
premier_depassement([3, 8], 10)         # renvoie None
```

```
# Votre code

# Tests
assert premier_depassement([3, 8, 12, 20], 10) == 2
assert premier_depassement([15, 8], 10) == 0
assert premier_depassement([3, 8], 10) == None
assert premier_depassement([], 10) == None
print("Tests OK")
```

3 Boucles for et listes

3.1 Exercice 7

Écrire une fonction `compte(t, x)` qui renvoie le nombre de fois où `x` apparaît dans la liste `t` (sans utiliser `t.count`).

```
compte([1, 3, 1, 2, 1], 1) # renvoie 3
```

```
# Votre code

# Tests
assert compte([1, 3, 1, 2, 1], 1) == 3
assert compte([1, 3, 1, 2, 1], 5) == 0
assert compte([], 1) == 0
assert compte(["a", "b", "a"], "a") == 2
print("Tests OK")
```

3.2 Exercice 8

Écrire une fonction `moyenne(notes)` qui renvoie la moyenne des nombres de la liste `notes` (sans `sum`), ou `None` si la liste est vide.

```
moyenne([12, 8, 16]) # renvoie 12.0
moyenne([])          # renvoie None
```

```
# Votre code

# Tests
assert moyenne([12, 8, 16]) == 12.0
assert moyenne([15]) == 15
assert moyenne([10, 11]) == 10.5
assert moyenne([]) == None
print("Tests OK")
```

3.3 Exercice 9

Écrire une fonction `pairs(t)` qui renvoie une **nouvelle** liste contenant uniquement les nombres pairs de `t`, dans le même ordre.

```
pairs([1, 4, 7, 8, 10]) # renvoie [4, 8, 10]
```

```
# Votre code
```

```
# Tests
t = [1, 4, 7, 8, 10]
assert pairs(t) == [4, 8, 10]
assert t == [1, 4, 7, 8, 10] # t n'a pas été modifiée
assert pairs([1, 3]) == []
assert pairs([]) == []
print("Tests OK")
```

3.4 Exercice 10

Écrire une fonction `position_max(t)` qui renvoie le couple (maximum, indice) du plus grand élément de la liste **non vide** `t`. En cas d'égalité, on garde le premier. Utiliser `enumerate`.

```
position_max([1, 5, 6, 9, 1, 9]) # renvoie (9, 3)
```

```
# Votre code
```

```
# Tests
assert position_max([1, 5, 6, 9, 1, 9]) == (9, 3)
assert position_max([7]) == (7, 0)
assert position_max([-4, -2, -8]) == (-2, 1)
print("Tests OK")
```

3.5 Exercice 11

Écrire une fonction `compte_voyelles(texte)` qui renvoie le nombre de voyelles (aeiouy) de la chaîne `texte`, écrite en minuscules.

Rappel : on peut parcourir une chaîne avec `for`, comme une liste.

```
compte_voyelles("bonjour") # renvoie 3
```

```
# Votre code
```

```
# Tests
assert compte_voyelles("bonjour") == 3
assert compte_voyelles("python") == 2
assert compte_voyelles("brr") == 0
assert compte_voyelles("") == 0
print("Tests OK")
```

4 Dictionnaires

4.1 Exercice 12

Écrire une fonction `total_stock(stock)` qui renvoie le nombre total d'articles d'un dictionnaire de type `{produit: quantité}`.

```
total_stock({"vis": 40, "clou": 0, "boulon": 12}) # renvoie 52
```

```
# Votre code
```

```
# Tests
assert total_stock({"vis": 40, "clou": 0, "boulon": 12}) == 52
assert total_stock({"vis": 3}) == 3
assert total_stock({}) == 0
print("Tests OK")
```

4.2 Exercice 13

Écrire une fonction `ruptures (stock)` qui renvoie la liste des produits dont la quantité est nulle.

```
ruptures({"vis": 40, "clou": 0, "boulon": 12, "ecrou": 0}) # renvoie ['clou', 'ecrou']
```

```
# Votre code
```

```
# Tests
assert ruptures({"vis": 40, "clou": 0, "boulon": 12, "ecrou": 0}) == ["clou", "ecrou"]
assert ruptures({"vis": 40}) == []
assert ruptures({}) == []
print("Tests OK")
```

4.3 Exercice 14

Écrire une fonction `occurrences (t)` qui renvoie un dictionnaire donnant, pour chaque élément de la liste `t`, son nombre d'apparitions.

```
occurrences(["a", "b", "a", "c", "a"]) # renvoie {'a': 3, 'b': 1, 'c': 1}
```

```
# Votre code
```

```
# Tests
assert occurrences(["a", "b", "a", "c", "a"]) == {"a": 3, "b": 1, "c": 1}
assert occurrences([2, 2, 2]) == {2: 3}
assert occurrences([]) == {}
print("Tests OK")
```

4.4 Exercice 15

Écrire une fonction `meilleur (notes)` qui prend un dictionnaire de type `{prénom: note}` et renvoie le prénom qui a la meilleure note (le premier en cas d'égalité), ou `None` si le dictionnaire est vide.

```
meilleur({"lea": 14, "tom": 17, "ana": 11}) # renvoie 'tom'
```

```
# Votre code
```

```
# Tests
assert meilleur({"lea": 14, "tom": 17, "ana": 11}) == "tom"
assert meilleur({"lea": 14}) == "lea"
assert meilleur({"lea": 15, "tom": 15}) == "lea"
assert meilleur({"lea": -1, "tom": -3}) == "lea"
assert meilleur({}) == None
print("Tests OK")
```