

MOOC Python

Corrigés de la semaine 4

dispatch1 - Semaine 4 Séquence 2

```
1 def dispatch1(a, b):
2     """dispatch1 comme spécifié"""
3     # si les deux arguments sont pairs
4     if a%2 == 0 and b%2 == 0:
5         return a*a + b*b
6     # si a est pair et b est impair
7     elif a%2 == 0 and b%2 != 0:
8         return a*(b-1)
9     # si a est impair et b est pair
10    elif a%2 != 0 and b%2 == 0:
11        return (a-1)*b
12    # sinon - c'est que a et b sont impairs
13    else:
14        return a*a - b*b
```

dispatch2 - Semaine 4 Séquence 2

```
1 def dispatch2(a, b, A, B):
2     """dispatch2 comme spécifié"""
3     # les deux cas de la diagonale \
4     if (a in A and b in B) or (a not in A and b not in B):
5         return a*a + b*b
6     # sinon si b n'est pas dans B
7     # ce qui alors implique que a est dans A
8     elif b not in B:
9         return a*(b-1)
10    # le dernier cas, on sait forcément que
11    # b est dans B et a n'est pas dans A
12    else:
13        return (a-1)*b
```

```

1 def libelle(ligne):
2     # on enlève les espaces et les tabulations
3     ligne = ligne.replace(' ', '').replace('\t','')
4     # on cherche les 3 champs
5     mots = ligne.split(',')
6     # si on n'a pas le bon nombre de champs
7     # rappelez-vous que 'return' tout court
8     # est équivalent à 'return None'
9     if len(mots) != 3:
10         return
11     # maintenant on a les trois valeurs
12     nom, prenom, rang = mots
13     # comment présenter le rang
14     rang_ieme = "1er" if rang == "1" \
15                 else "2nd" if rang == "2" \
16                 else f"{rang}-ème"
17     return f"{prenom}.{nom} ({rang_ieme})"

```

```

1 def pgcd(a, b):
2     # le cas pathologique
3     if a * b == 0:
4         return 0
5     "le pgcd de a et b par l'algorithme d'Euclide"
6     # l'algorithme suppose que a >= b
7     # donc si ce n'est pas le cas
8     # il faut inverser les deux entrées
9     if b > a :
10         a, b = b, a
11     # boucle sans fin
12     while True:
13         # on calcule le reste
14         r = a % b
15         # si le reste est nul, on a terminé
16         if r == 0:
17             return b
18         # sinon on passe à l'itération suivante
19         a, b = b, r

```

pgcd (bis) - Semaine 4 Séquence 3

```
1  # il se trouve qu'en fait la première inversion n'est
2  # pas nécessaire
3  # en effet si a <= b, la première itération de la boucle
4  # while va faire
5  # r = a % b = a
6  # et ensuite
7  # a, b = b, r = b, a
8  # ce qui provoque l'inversion
9  def pgcd_bis(a, b):
10     # le cas pathologique
11     if a == 0 or b == 0:
12         return 0
13     while True:
14         # on calcule le reste
15         r = a % b
16         # si le reste est nul, on a terminé
17         if r == 0:
18             return b
19         # sinon on passe à l'itération suivante
20         a, b = b, r
```

pgcd (ter) - Semaine 4 Séquence 3

```
1  # une autre alternative, qui fonctionne aussi
2  # plus court, mais on passe du temps à se convaincre
3  # que ça fonctionne bien comme demandé
4  def pgcd_ter(a, b):
5     # le cas pathologique
6     if a * b == 0:
7         return 0
8     # si on n'aime pas les boucles sans fin
9     # on peut faire aussi comme ceci
10     while b:
11         a, b = b, a % b
12     return a
```

```

1 import math
2
3 def distance(*args):
4     "la racine de la somme des carrés des arguments"
5     # avec une compréhension on calcule la liste des carrés des arguments
6     # on applique ensuite sum pour en faire la somme
7     # vous pourrez d'ailleurs vérifier que sum ([]) = 0
8     # enfin on extrait la racine avec math.sqrt
9     return math.sqrt(sum([x**2 for x in args]))

```

```

1 from operator import mul
2
3 def numbers(*liste):
4     """
5     retourne un tuple contenant
6     (*) la somme
7     (*) le minimum
8     (*) le maximum
9     des éléments de la liste
10    """
11
12    if not liste:
13        return 0, 0, 0
14
15    return (
16        # la builtin 'sum' renvoie la somme
17        sum(liste),
18        # les builtin 'min' et 'max' font ce qu'on veut aussi
19        min(liste),
20        max(liste),
21    )

```

```
1  # en regardant bien la documentation de sum, max et min,
2  # on voit qu'on peut aussi traiter le cas singulier
3  # (pas d'argument) en passant
4  #   start à sum
5  #   et default à min ou max
6  # comme ceci
7  def numbers_bis(*liste):
8      return (
9          # attention:
10         # la signature de sum est: sum(iterable[, start])
11         # du coup on ne peut pas passer à sum start=0
12         # parce que start n'a pas de valeur par défaut
13         sum(liste, 0),
14         # par contre avec min c'est min(iterable, *[, key, default])
15         # du coup on doit appeler min avec default=0 qui est plus clair
16         # l'étoile qui apparaît dans la signature
17         # rend le paramètre default keyword-only
18         min(liste, default=0),
19         max(liste, default=0),
20     )
```