

MOOC Python 3

Session 2018

Corrigés de la semaine 4

dispatch1 - Semaine 4 Séquence 2

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

```

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

```

```

1 def libelle(ligne):
2     """
3     n'oubliez pas votre docstring
4     """
5     # on cherche les 3 champs après avoir nettoyé
6     # les éléments séparés par une virgule
7     mots = [mot.strip() for mot in ligne.split(',')]
8     # si on n'a pas le bon nombre de champs
9     # rappelez-vous que 'return' tout court
10    # est équivalent à 'return None'
11    if len(mots) != 3:
12        return
13    # maintenant on a les trois valeurs
14    nom, prenom, rang = mots
15    # comment présenter le rang
16    rang_ieme = "1er" if rang == "1" \
17                else "2nd" if rang == "2" \
18                else f"{rang}-ème"
19    return f"{prenom}.{nom} ({rang_ieme})"

```

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

```

1 def pgcd_bis(a, b):
2     """
3     Il se trouve qu'en fait la première
4     inversion n'est pas nécessaire.
5
6     En effet si a <= b, la première itération
7     de la boucle while va faire:
8     reste = a % b c'est-à-dire a
9     et ensuite
10    a, b = b, reste = b, a
11    provoque l'inversion
12    """
13    # si l'un des deux est nul on retourne l'autre
14    if a * b == 0:
15        return a or b
16    # sinon on fait une boucle sans fin
17    while True:
18        # on calcule le reste
19        reste = a % b
20        # si le reste est nul, on a terminé
21        if reste == 0:
22            return b
23        # sinon on passe à l'itération suivante
24        a, b = b, reste

```

```

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

```

```
1  # une solution très élégante proposée par adrienollier
2
3  # les tranches en ordre décroissant
4  bareme = (
5      (150_000, 45),
6      (50_000, 40),
7      (12_500, 20),
8      (0, 0),
9  )
10
11 def taxes(revenu):
12     """
13     U.K. income taxes calculator
14     https://www.gov.uk/income-tax-rates
15     """
16     montant = 0
17     for seuil, taux in bareme:
18         if revenu > seuil:
19             montant += (revenu - seuil) * taux // 100
20             revenu = seuil
21     return montant
```

```

1
2 # cette solution est plus pataude; je la retiens
3 # parce qu'elle montre un cas de for .. else ..
4 # qui ne soit pas trop tiré par les cheveux
5 # quoique
6
7 bands = [
8     # à partir de 0. le taux est nul
9     (0, 0.),
10    # jusqu'à 12 500 où il devient de 20%
11    (12_500, 20/100),
12    # etc.
13    (50_000, 40/100),
14    (150_000, 45/100),
15 ]
16
17 def taxes_bis(income):
18     """
19     Utilise un for avec un else
20     """
21     amount = 0
22
23     # en faisant ce zip un peu étrange, on va
24     # considérer les couples de tuples consécutifs dans
25     # la liste bands
26     for (band1, rate1), (band2, _) in zip(bands, bands[1:]):
27         # le salaire est au-delà de cette tranche
28         if income >= band2:
29             amount += (band2-band1) * rate1
30         # le salaire est dans cette tranche
31         else:
32             amount += (income-band1) * rate1
33             # du coup on peut sortir du for par un break
34             # et on ne passera pas par le else du for
35             break
36     # on ne passe ici qu'avec les salaires dans la dernière tranche
37     # en effet pour les autres on est sorti du for par un break
38     else:
39         band_top, rate_top = bands[-1]
40         amount += (income - band_top) * rate_top
41     return int(amount)

```

```
1 def int_to_char(n):
2     """
3     traduit un entier entre 1 et 26
4     en un caractère entre 'A' et 'Z'
5     """
6     # si index était compris entre 0 et 25, on pourrait obtenir
7     # la lettre comme étant chr(ord('A') + index)
8     # on fait donc un changement de variable n -> n-1
9     # de plus on va rendre le résultat cyclique modulo 26
10    # pour pouvoir l'utiliser sur des nombres quelconques
11
12    return chr(ord('A') + (n - 1) % 26)
13
14
15 def spreadsheet(index):
16     """
17     transforme un numéro de colonne en nom alphabétique
18     dans l'ordre lexicographique
19     1 -> A; 26 -> Z; 27 -> AA; 28 -> AB; etc..
20     """
21     # index peut être supérieur à 26
22     # en remarquant que la dernière lettre s'incrémente à chaque fois
23     # qu'index augmente, et repasse à 'A' de manière cyclique,
24     # on voit qu'on peut utiliser notre version cyclique de `int_to_char`
25     # pour calculer la lettre la plus à droite dans le résultat.
26     # et pour les autres lettres, il suffit de recommencer sur le quotient
27
28     result = int_to_char(index)
29     while index > 26:
30         index = (index - 1) // 26
31         result = int_to_char(index) + result
32     return result
```

```
1 def spreadsheet_bis(index):
2     """
3     Accessoirement on peut vérifier que la variable index fournie
4     est bien un entier supérieur à 0.
5     """
6     if not isinstance(index, int):
7         raise TypeError("index must be an integer !")
8     elif index < 1:
9         raise ValueError("index must be positive !")
10
11     result = chr(ord('A') + (index - 1) % 26)
12     while index > 26:
13         index = (index - 1) // 26
14         result = chr(ord('A') + (index - 1) % 26) + result
15     return result
```



```

1  # la fonction int_to_char n'a pas besoin d'être exposée
2  # dans l'espace de nommage du module.
3  # puisque c'est une fonction assistante,
4  # on peut en faire une variable locale à spreadsheet_ter
5  # en la déclarant à l'intérieur de la fonction
6  def spreadsheet_ter(index):
7      """
8      transforme un numéro de colonne en nom alphabétique
9      dans l'ordre lexicographique
10     1 -> A; 26 -> Z; 27 -> AA; 28 -> AB; etc..
11     """
12     def int_to_char(n):
13         """
14         traduit un entier entre 1 et 26
15         en un caractère entre 'A' et 'Z'
16         """
17         return chr(ord('A') + (n - 1) % 26)
18
19     if not isinstance(index, int):
20         raise TypeError("index must be an integer!")
21     elif index < 1:
22         raise ValueError("index must be positive!")
23
24     # ici int_to_char est une variable locale
25     # à la fonction spreadsheet_ter
26     result = int_to_char(index)
27     while index > 26:
28         index = (index - 1) // 26
29         # idem ici bien sûr
30         result = int_to_char(index) + result
31     return result

```

```

1 def power(x, n):
2     """
3     mise à la puissance en  $O(\log_2(n))$ 
4     """
5     # on s'astreint à ne pas utiliser ** parce que ce serait triché
6     # mais bien sûr dans la pratique
7     # on pourrait utiliser **2 pour traiter le cas où n est pair
8     if n == 1:
9         return x
10    elif n % 2 == 0:
11        # on met au carré power(x, n//2)
12        # une petite subtilité ici, c'est que si vous écrivez
13        # root = power(x, n//2) * power(x, n//2)
14        # vous allez évaluez **deux fois** power()
15        # et du coup vous perdez tout le bénéfice de l'exercice
16        root = power(x, n//2)
17        return root * root
18    else:
19        return x * power(x, n-1)

```

```

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

```

```

1 def distance_bis(*args):
2     """
3     Idem mais avec une expression génératrice
4     """
5     # on n'a pas encore vu cette forme - cf Semaine 5
6     # mais pour vous donner un avant-goût d'une expression
7     # génératrice:
8     # on peut faire aussi comme ceci
9     # observez l'absence de crochets []
10    # la différence c'est juste qu'on ne
11    # construit pas la liste des carrés,
12    # car on n'en a pas besoin
13    # et donc un itérateur nous suffit
14    return math.sqrt(sum(x**2 for x in args))

```

```

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

```

```
1  # en regardant bien la documentation de sum, max et min,
2  # on voit qu'on peut aussi traiter le cas singulier
3  # (où il n'y 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, la signature de sum est:
10         #   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         # on pourrait par contre faire juste sum(liste)
14         # car le défaut pour start c'est 0
15         # dit autrement, sum([]) retourne bien 0
16         sum(liste, 0),
17         # par contre avec min c'est
18         #   min(iterable, *[, key, default])
19         # du coup on DOIT appeler min avec default=0 qui est plus clair
20         # l'étoile qui apparaît dans la signature
21         # rend le paramètre default keyword-only
22         min(liste, default=0),
23         max(liste, default=0),
24     )
```