MOOC Python 3

Session 2018

Corrigés de la semaine 4

dispatch1 - Semaine 4 Séquence 2

```
def dispatch1(a, b):
    """
    dispatch1 comme spécifié
    """
    # si les deux arguments sont pairs
    if a%2 == 0 and b%2 == 0:
        return a*a + b*b
    # si a est pair et b est impair
    elif a%2 == 0 and b%2 != 0:
        return a*(b-1)
    # si a est impair et b est pair
    elif a%2 != 0 and b%2 == 0:
        return (a-1)*b
    # sinon - c'est que a et b sont impairs
    else:
        return a*a - b*b
```

dispatch2 - Semaine 4 Séquence 2

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

libelle - Semaine 4 Séquence 2

```
def libelle(ligne):
    """
    n'oubliez pas votre docstring
    """
    # on enlève les espaces et les tabulations
    ligne = ligne.replace(' ', '').replace('\t','')
    # on cherche les 3 champs
    mots = ligne.split(',')
    # si on n'a pas le bon nombre de champs
    # rappelez-vous que 'return' tout court
    # est équivalent à 'return None'
    if len(mots) != 3:
        return
    # maintenant on a les trois valeurs
    nom, prenom, rang = mots
    # comment présenter le rang
    rang_ieme = "1er" if rang == "1" \
                else "2nd" if rang == "2" \
                else f"{rang}-ème"
    return f"{prenom}.{nom} ({rang_ieme})"
```

pgcd - Semaine 4 Séquence 3

```
def pgcd(a, b):
    """
    le pgcd de a et b par l'algorithme d'Euclide
    """
    # l'algorithme suppose que a >= b
    # donc si ce n'est pas le cas
    # il faut inverser les deux entrées
    if b > a:
        a, b = b, a
    if b == 0:
        return a
    # boucle sans fin
    while True:
        # on calcule le reste
        reste = a % b
        # si le reste est nul, on a terminé
        if reste == 0:
            return b
        # sinon on passe à l'itération suivante
        a, b = b, reste
```

pgcd (bis) - Semaine 4 Séquence 3

```
def pgcd_bis(a, b):
    """
    Il se trouve qu'en fait la première
    inversion n'est pas nécessaire.

    En effet si a <= b, la première itération
    de la boucle while va faire:
    reste = a % b  c'est-à-dire a
    et ensuite
    a, b = b, reste = b, a
    provoque l'inversion
    """
    # si l'on des deux est nul on retourne l'autre
    if a * b == 0:
        return a or b
    # sinon on fait une boucle sans fin
    while True:
        # on calcule le reste
        reste = a % b
        # si le reste est nul, on a terminé
        if reste == 0:
            return b
        # sinon on passe à l'itération suivante
        a, b = b, reste
```

pgcd (ter) - Semaine 4 Séquence 3

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

taxes - Semaine 4 Séquence 3

```
# une solution très élégante proposée par adrienollier

# les tranches en ordre décroissant
TaxRate = (
    (150_000, 45),
    (45_000, 40),
    (11_500, 20),
    (0, 0),
)

def taxes(income):
    """
    U.K. income taxes calculator
    https://www.gov.uk/income-tax-rates
    """
    due = 0
    for floor, rate in TaxRate:
        if income > floor:
            due += (income - floor) * rate / 100
            income = floor
    return int(due)
```

taxes (bis) - Semaine 4 Séquence 3

```
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 11 500 où il devient de 20%
11      (11_500, 20/100),
12      # etc.
13      (45_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)
```

distance - Semaine 4 Séquence 6

```
import math

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

distance (bis) - Semaine 4 Séquence 6

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

numbers - Semaine 4 Séquence 6

```
def numbers(*liste):
    """
    retourne un tuple contenant
    (*) la somme
    (*) le minimum
    (*) le maximum
    des éléments de la liste
    """

    if not liste:
        return 0, 0, 0

    return (
        # la builtin 'sum' renvoie la somme
        sum(liste),
        # les builtin 'min' et 'max' font ce qu'on veut aussi
        min(liste),
        max(liste),
    )
```

numbers (bis) - Semaine 4 Séquence 6

```
# en regardant bien la documentation de sum, max et min,
# on voit qu'on peut aussi traiter le cas singulier
# (où il n'y pas d'argument) en passant
#   start à sum
#   et default à min ou max
# comme ceci
def numbers_bis(*liste):
    return (
        # attention, la signature de sum est:
        #   sum(iterable[, start])
        # du coup on ne PEUT PAS passer à sum start=0
        # parce que start n'a pas de valeur par défaut
        # on pourrait par contre faire juste sum(liste)
        # car le défaut pour start c'est 0
        # dit autrement, sum([]) retourne bien 0
        sum(liste, 0),
        # par contre avec min c'est
        #  min(iterable, *[, key, default])
        # du coup on DOIT appeler min avec default=0 qui est plus clair
        # l'étoile qui apparaît dans la signature
        # rend le paramètre default keyword-only
        min(liste, default=0),
        max(liste, default=0),
    )
```