# MOOC Python

# 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):
    # 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 cas pathologique
    if a * b == 0:
        return 0
    "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
    # boucle sans fin
    while True:
        # on calcule le reste
        r = a % b
        # si le reste est nul, on a terminé
        if r == 0:
            return b
        # sinon on passe à l'itération suivante
        a, b = b, r
```

pgcd (bis) - Semaine 4 Séquence 3

```
# 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
# r = a % b = a
# et ensuite
# a, b = b, r = b, a
# ce qui provoque l'inversion
def pgcd_bis(a, b):
    # le cas pathologique
    if a == 0 or b == 0:
        return 0
    while True:
        # on calcule le reste
        r = a % b
        # si le reste est nul, on a terminé
        if r == 0:
            return b
        # sinon on passe à l'itération suivante
        a, b = b, r
```

pgcd (ter) - Semaine 4 Séquence 3

```
# une autre alternative, qui fonctionne aussi
# plus court, mais on passe du temps à se convaincre
# que ça fonctionne bien comme demandé
def pgcd_ter(a, b):
    # le cas pathologique
    if a * b == 0:
        return 0
    # 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

```
# la définition des différentes tranches
bands = [
    # à partir de 0. le taux est nul
    (0, 0.),
    # jusqu'à 11 500 où il devient de 20%
    (11_500, 20/100),
    # etc.
    (45_000, 40/100),
    (150_000, 45/100),
]


def taxes(income):
    """
    calcule l'impôt sur le revenu
    en U.K. selon le barême
    https://www.gov.uk/income-tax-rates

    utilise un for avec un break
    """
    # on accumule les morceaux
    amount = 0
    # en faisant ce zip un peu étrange, on va
    # considérer les couples de tuples consécutifs dans
    # la liste bands
    for (b1, rate1), (b2, _) in zip(bands, bands[1:]):
        #print(f"{b1:6} {b2:6}", end=' ')
        # le salaire est au-delà de cette tranche
        if income >= b2:
            delta = (b2-b1) * rate1
            #print(f"(1) base = {b2-b1}, rate = {rate1} -> {delta}")
            amount += delta
        # le salaire est dans cette tranche
        else:
            delta = (income-b1) * rate1
            #print(f"(2) base = {income-b1}, rate = {rate1} -> {delta}")
            amount += delta
            # du coup on peut sortir du for par un break
            # et on ne passera pas par le else du for
            break
    # on ne passe ici qu'avec les salaires dans la dernière tranche
    # en effet pour les autres on est sorti du for par un break
    else:
        btop, rate_top = bands[-1]
        #print(f"{btop:6} {6*'.'}", end=' ')
        delta = (income - btop) * rate_top
        #print(f"(3) base = {income-btop}, rate = {rate1} -> {delta}")
        amount += delta
    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]))
```

numbers - Semaine 4 Séquence 6

```
from operator import mul

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
# (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
        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),
    )
```