ex 0

```
print("Hello World!")
```

ex 1.1

```
message = "Je connais la réponse à l'univers, la vie et le reste"
reponse = 6 * 7

print(message)
print(reponse)
```

ex1.2

```
a = 567 * 72
b = 33**4
c = 98.2 / 6
d = (7 * 9)**4 / 6

print(a)
print(b)
print(c)
print(d)
```

ex 2.1

```
age_string = input('Quel est votre age ?\n')
age_int = int(age_string)

print("Dans deux ans vous aurez")
print(age_int+2)
```

ex 2.2

```
annee = int(input('Quelle est votre année de naissance ?\n'))
age = 2018 - annee + 2
print("Dans deux ans vous aurez {} ans.".format(age))
```

3.1 Compter les lettres

```
mot = input("Donnez moi un mot.\n")
print("Ce mot fait {} caractères (espaces inclus).".format(len(mot)))
```

3.2 Changer des lettres

```
mot_modifie = mot.replace('a', 'z').replace('e', 'y')
print("Mot modifié : {}".format(mot_modifie))
```

3.3 Encadrer le mot avec

```
mot_encadre = '#### ' + mot + ' ####'
print("mot encadré: {}".format(mot_encadre))
```

4. fonctions de centrage

```
def centrer(mot, largeur, caractere):

    longueur_part_1 = len(mot) // 2 # petite moitiée du mot si longeur impaire
    longueur_part_2 = len(mot) - longueur_part_1 # grande moitiée du mot si longueur impair

    decalage1 = (largeur // 2) - longueur_part_1 - 1
    decalage2 = (largeur - largeur // 2) - longueur_part_2 - 1

    premiere_ligne = caractere * largeur
    deuxieme_ligne = caractere +  (decalage1 * ' ') + mot + (decalage2 * ' ') + caractere

    resultat = premiere_ligne + '\n' + deuxieme_ligne + '\n' + premiere_ligne

    return resultat


print('Résultat:')
print(centrer('Pikachu', 41, '@')) # affiche le mot centré sur 80 caractère
print("Longueur du résultat (les sauts de lignes sont comptés car ce sont des charactères)"
print(len(centrer('Pikachu', 41, '@'))) # affiche la longueur du résultat pour vérifier qu'
```

5. conditions

```
def centrer(mot, largeur, caractere):
```

```
    caractere_vide = (not caractere)

    if caractere_vide:
        caractere = '|'

    if largeur == -1:
        ligne_centrage = "{} {} {}".format(caractere, mot, caractere)
        ligne_pleine = len(ligne_centrage) * caractere

    elif len(mot) + 4 > largeur:
        mot_affiche = mot[0:largeur-4]
        ligne_centrage = "{} {} {}".format(caractere, mot_affiche, caractere)
        ligne_pleine = len(ligne_centrage) * caractere

    else:
        longueur_part_1 = len(mot) // 2 # petite moitiée du mot si longeur impaire
        longueur_part_2 = len(mot) - longueur_part_1 # grande moitiée du mot si longueur imp

        decalage1 = (largeur // 2) - longueur_part_1 - 1
        decalage2 = (largeur - largeur // 2) - longueur_part_2 - 1

        ligne_pleine = caractere * largeur
        ligne_centrage = caractere +  (decalage1 * ' ') + mot + (decalage2 * ' ') + caracter

    if caractere_vide:
        resultat = ligne_centrage
    else:
        resultat = ligne_pleine + '\n' + ligne_centrage + '\n' + ligne_pleine

    return resultat

print('Résultat pour Pikachu, 41, @:')
print(centrer('Pikachu', 41, '@'))
print('\nRésultat pour Pikachu, -1, @:')
print(centrer('Pikachu', -1, '@'))
print("\nRésultat pour Pikachu, 41, '':")
print(centrer('Pikachu', 41, ''))
print("\nRésultat pour Pikachu, 8, '#':")
print(centrer('Pikachu', 8, '#'))
```

6.1

```
def demander_nombre_pair():
    try:
        entier = int(input('Donner moi un entier pair :\n')) # ValueError si l'entrée n'est
```

```
        assert entier % 2 == 0 # AssertionError si l'entrée n'est pas un nombre pair
    except:
        raise Exception("Ce n'est pas un entier pair !")

    return entier

if __name__ == '__main__':
    print('résultat: {}'.format(demander_nombre_pair()))
```

6.2

```
def centrer(mot, largeur, caractere):

    assert largeur > 0 or largeur == -1
    assert len(caractere) == 1 or caractere == ''
    assert isinstance(mot, str)

    caractere_vide = (not caractere)

    if caractere_vide:
        caractere = '|'

    if largeur == -1:
        ligne_centrage = "{} {} {}".format(caractere, mot, caractere)
        ligne_pleine = len(ligne_centrage) * caractere

    elif len(mot) + 4 > largeur:
        mot_affiche = mot[0:largeur-4]
        ligne_centrage = "{} {} {}".format(caractere, mot_affiche, caractere)
        ligne_pleine = len(ligne_centrage) * caractere

    else:
        longueur_part_1 = len(mot) // 2 # petite moitiée du mot si longeur impaire
        longueur_part_2 = len(mot) - longueur_part_1 # grande moitiée du mot si longeur imp

        decalage1 = (largeur // 2) - longueur_part_1 - 1
        decalage2 = (largeur - largeur // 2) - longueur_part_2 - 1

        ligne_pleine = caractere * largeur
        ligne_centrage = caractere +  (decalage1 * ' ') + mot + (decalage2 * ' ') + caracter

    if caractere_vide:
        resultat = ligne_centrage
    else:
        resultat = ligne_pleine + '\n' + ligne_centrage + '\n' + ligne_pleine
```

```
    return resultat

# print(centrer('Pikachu', -2, '@')) #
print(centrer('Pikachu', 41, '@@'))
# print(centrer(-2, 41, ''))
```

7.1, 7.2, 7.3, 7.4

```
def table_de_multiplication(nombre):

    assert isinstance(nombre, int)

    resultat = '| Table de {} \n--------------\n'.format(nombre)

    for n in range(1,11):
        resultat += '| {} x {} = {} \n'.format(n, nombre, n * nombre)

    return resultat

if __name__ == '__main__':

    while True:
        mdp = input('Entrez le mot de passe\n')

        if mdp == 'pyth0ng1t':
            for n in range(1, 11):
                print(table_de_multiplication(n))

            break
```

7.5

```
def is_prime(nombre):
    for n in range(3, nombre):
        if nombre % n == 0:
            return False
    return True
```

7.6

def fibo(maximum): fib_a = 1 fib_b = 1 result = '{}'.format(fib_a) for n in range(1, maximum): result += '{}'.format(fib_b) fib_c = fib_a + fib_b fib_a = fib_b fib_b = fib_c return result

if **name** == '**main**': print(is_prime(73)) print(is_prime(60))

```
n = int(input('nombre de terme pour fibonacci:\n'))
print(fibo(n))
```

‘‘‘

7.7

```
def afficher_allumettes(n):
    allumettes = ' ' + n * '|'
    print(allumettes)

def choisir_nombre():
    while True:
        try:
            nombre = int(input("Choisir un nombre entre 1 et 3 :\n"))
            assert 1 <= nombre <= 3
        except (ValueError, AssertionError): # attention à ne pas catcher toutes les except
            continue
        return nombre

def partie_allumettes(total):
    while total > 0:
        afficher_allumettes(total)
        n = 0
        while True:
            n = choisir_nombre()
            if total - n < 0:
                continue
            break
        total -= n
```

7.8

```
def play(allumettes_restantes):
    if 9 > allumettes_restantes > 5:
        return allumettes_restantes - 5
```

```
    elif allumettes_restantes == 5:
        return 2
    elif 5 > allumettes_restantes > 1:
        return allumettes_restantes - 1
    else:
        return 1


partie_allumettes(17)
```

8.1.1, 8.1.2

```
def plus_grand(l):
    try:
        assert l
        assert isinstance(l, list)
    except AssertionError:
        raise Exception("le parametre de plus_grand doit être une liste non vide")

    plus_grand = l[0]
    for n in l:
        if n > plus_grand:
            plus_grand = n
    return plus_grand
```

8.2

```
def somme(l):
    total = 0
    for n in l:
        total += n
    return total
```

8.3

```
def somme_recursive(l):
    if len(l) == 1:
        return l[0]
    else:
        return l[0] + somme_recursive(l[1:])
```

8.4

```python
def elem_pairs(l):
    return [elem for elem in l if elem % 2 == 0]
```

8.5

```python
# tri par insertion
def TriInsertion(liste):
    for i in range(1, len(liste)): # on parcoure la liste de gauche à droite
        pivot = liste[i] # pivot = nombre courant à ranger ; on le sauvegarde
        j = i
        while j > 0 and liste[j - 1] > pivot: # j parcoure les cases vers la gauche. Tant qu
            liste[j] = liste[j - 1] # ... on décale les nombres de gauche (déjà triés) vers
            j -= 1
        liste[j] = pivot # quand on a trouvé la place du pivot on l'insère
    return liste


print(plus_grand([5, 9, 12, 6, -1, 4]))
# print(plus_grand([]))
print(somme([1,9,4,8]))
print(somme_recursive([1,9,4,8]))
print(TriInsertion([4,6,3,5,9,1,7]))
```