Ex2

October 1, 2019

1 Algorithmique et Pensée Computationnelle

2 Introduction à Python

2.0.1 A. Fonction input

Demande input d'utilisateur

Completez le code afin que le contenu de input_string soit l'output.

[ ]:
```
input_string = input("Quel est votre prénom? ")
print("Bonjour, ", input_string)
```

2.0.2 Exercice:

Modifiez le code ci dessus pour également demander le nom de famille puis affichez `Bonjour, {prénom} {nom}`

2.0.3 B. Variables

Définition de variable, type de variable, conversion de type, opérateurs arithmétiques, opérateurs booléens et opérateurs de comparaison.

Introduisez une variable de type string nom_chien, une variable de type integer age_chien.

Affichez "Mon chien nommé (nom_chien) et agé de (age_chien) ans." en utilisant la fonction format().

Affichez "Mon chien nommé (nom_chien) et agé de (age_chien) ans." en utilisant le symbole %.

[ ]:
```
nom_chien = "Lola"
age_chien = 3

print("Mon chien nommé {0} et agé de {1} ans.".format(nom_chien,age_chien))
print("Mon chien nommé %s et agé de %d ans." % (nom_chien,age_chien))
```

2.0.4 Exercice:

Changez le code ci dessus pour demander à l'utilisateur d'entrer le nom de son chien

[ ]:
```
nom_chien = "Lola"
age_chien = 3

print(type(nom_chien))
print(type(age_chien))
```

2.0.5 Exercice:

Modifiez la valeur de la variable `age_chien` afin que le chiffre soit un float (nombre à virgule).

Assurez vous que la variable `age_chien` soit un float en affichant le type de la variable.

Affichez la valeur de la variable `age_chien` en float et en integer.

Vous pouvez vérifier le type de chaque variable dans la cellule ci dessus

2.0.6 Exercice:

Créez une variable `poids_chien`, `resultat`, `remainder`.

Divisez la valeur de la variable `poids_chien` par deux et enregistrez le résultat dans la variable `resultat`.

Trouvez le reste de la division grâce à la fonction modulo `%` et enregistrez le résultat dans la variable `remainder`.

[ ]:
```
from assertion import assert_division

# VOTRE CODE ICI
poids_chien =
poids_ideal =
resultat =
remainder =
# FIN DE VOTRE CODE

assert_division(poids_chien, resultat, remainder)
```

Diminuez la valeur de la variable `poids_ideal` de 2.

Diminuez la valeur de la variable `poids_chien` de 1 .

[ ]:
```
from assertion import State, assert_sub
prev = State("sub", poids_chien=poids_chien, poids_ideal=poids_ideal)

# VOTRE CODE ICI
poids_ideal -=
```

```
poids_chien -=
# FIN DE VOTRE CODE

assert_sub(prev, State(poids_chien=poids_chien, poids_ideal=poids_ideal))
```

2.0.7 Exercice:

Vérifiez si la valeur de la variable `poids_chien` est égale à celle de la variable `poids_ideal` en utilisant un opérateur qui donne une réponse booléenne `True` ou `False`. Stockez la réponse booléenne dans la variable `egalite_poids`.

Affichez la valeur de la variable `egalite_poids`.

[ ]:
```
from assertion import assert_equal


# VOTRE CODE ICI
egalite_poids =
# FIN DE VOTRE CODE

assert_equal(poids_chien, poids_ideal, egalite_poids)
```

2.0.8 Exercice:

Vérifiez que la valeur de la variable `poids_chien` est plus petite ou égale que celle de la variable `poids_ideal`. Stockez la réponse booléenne dans la variable `comparaison_poids`.

Affichez la valeur de la variable `comparaison_poids`.

Vérifiez que la valeur de la variable `poids_ideal` est plus grande que celle de la variable `poids_chien` en une seule ligne grâce à la fonction `print`.

[ ]:
```
from assertion import assert_comparison


# VOTRE CODE ICI
comparison_poids =
# FIN DE VOTRE CODE

assert_comparison(poids_chien, poids_ideal, comparison_poids)
```

2.0.9 C. Strings

Index, slicing, in, length

Définissez une nouvelle variable du nom `python3` de type `string` contenant `"Hard but cool!"` ou un autre texte de votre choix (d'au moins 13 charactères).

Definissez la valeur de la variable `premiere_lettre` comme le premier charactère de `python3` en utilisant l'indexation.

Definissez la valeur de la variable `derniere_lettre` comme le dernier charactère de `python3` en utilisant l'indexation.

Affichez la valeur de `premiere_lettre` et `derniere_lettre` sur deux lignes disctintes.

[ ]:
```
from assertion import assert_strings

# VOTRE CODE ICI
python3 =
premiere_lettre =
derniere_lettre =
# FIN DE VOTRE CODE

assert_strings(python3, premiere_lettre, derniere_lettre)
```

2.0.10 Exercice:

Définissez une variable `quatre` qui reprend les `4` premiers charactères de `python3` en utilisant le slicing `str[ind1:ind2]`.

Définissez une variable `neuf_treize` qui reprend les charactères `9 à 13` de `python3` en utilisant le slicing `str[ind1:ind2]`.

Définissez une variable `cinq_huit` qui reprend le "but" de `python3` en utilisant le slicing `str[ind1:ind2]`.

Affichez les valeurs de `quatre`, `neuf_treize` et `cinq_huit` sur une seule ligne.

[ ]:
```
from assertion import assert_slicing

# VOTRE CODE ICI
quatre =
neuf_treize =
cinq_huit =
# FIN DE VOTRE CODE

assert_slicing(python3, quatre,neuf_treize,cinq_huit)
```

Vérifiez et affichez que le mot "joke" se trouve dans `python3` grâce à l'opérateur `in`.

Vérifiez et affichez que le mot "but" se trouve dans `python3` grâce à l'opérateur `in`.

[ ]:
```
print("joke" in python3)
print("but" in python3)
```

2.0.11 Exercice:

Définissez `first_half` comme étant la première moitié de charactères stocké dans python3 en utilisant la fonction `len()`.

Affichez `first_half`.

[ ]:
```
from assertion import assert_half

# VOTRE CODE ICI
first_half =
# FIN DE VOTRE CODE

print(first_half)

assert_half(python3, first_half)
```

2.0.12 Exercice:

Définissez et affichez `dont_worry` comme "Don't worry, you're gonna become a king!"

Affichez "It's just like Rock'n'Roll!" sans créer de nouvelle variable.

Affichez `dont_worry` en majuscule et minuscule sur deux lignes.

[ ]:
```
from assertion import assert_dont_worry

# VOTRE CODE ICI
dont_worry =
upper =
lower =
# FIN DE VOTRE CODE

assert_dont_worry(dont_worry, upper, lower)
```

2.0.13 D. Conditions

Opérateurs booléens, If, else

Définissez une variable string `name`, une variable integer `age`, une variable integer `brothers` et une variable `sisters`.

Vérifiez et affichez que `name` est égal à la valeur de `name` OU que `age` est égal à la valeur de `age` en une seule ligne. Faites la même chose avec AND NOT.

Vérifiez et affichez que `name` est égal à "Miro" OU PAS que `age` est supérieur à 5 en une seule ligne.

Affichez l'ordre de priorité des opérateurs booléens.

Affichez “La condition est vraie, donc le code peut être exécuter!” si `age` est supérieur à 5 ET longueur de `name` inférieur à 10. Utilisez if et l'indentation.

Affichez “I'm not alone” si `brothers` OU `sisters` supérieur à 1. Affichez “We have boys at home” si `brothers` supérieur à 1 ET `sisters` égal à 0. Affichez “We have girls at home” si `brothers` égal à 0 ET `sisters` supérieur à 1. Sinon affichez “I have a little goat and a skateboard”.

[ ]:
```
name="Paulina"
age=9
brothers=2
sisters=0
print(name=="Paulina" or age==9)
print(name=="Paulina" and not age==9)
print(name=="Miro" or not age>5)
print("Not, And, Or")
if age>5 and len(name)<10:
  print("La condition est vraie, donc le code peut être exécuter!")

if brothers>1 or sisters>1:
  print("I'm not alone")
elif brothers>1 and sisters==0:
  print("We have boys at home")
elif brothers==0 and sisters>1:
  print("We have girls at home")
else:
  print("I have a little goat and a skateboard")
```

2.0.14 F. Fonctions

Définition, appel, return, paramètre par défaut

Définissez une fonction nommée `vin` qui affiche “Neuchâtel?” une fois que la fonction est appelée.

Appelez la fonction `vin`.

Pour chaque `i` de rang 3, appelez la fonction `vin`.

[ ]:
```
def vin():
  print("Neuchâtel?")
vin()
for i in range(3):
  vin()
```

Définissez une fonction `multip` ayant deux paramètres `a` et `b`. La fonction retourne le résultat de la mutiplication `a*b`.

Créez un variable `result` qui appelle la fonction `multip` de `3` et `7`.

Affichez `result`.

[ ]:
```
def multip(a,b):
    return a*b
result=multip(3,7)
print(result)
```