Formulaire Python :

Types de variables

Variable a	Type(a)
a = 2	# int (nombre entier)
a = 1.5	# float (nombre décimal)
a = True	# bool (booléen)
a = "Hello !"	# str (chaine de caractères)
a = ("a","b")	# tuple (n-uplet)
a = [4,-5,64,11]	# list (tableau)
a = {clé1 : valeur, clé2 : valeur}	# dict (dictionnaire)

Tests entre deux variables

==	égalité
!=	différence
<=	inférieur ou égal
>=	supérieur ou égal
nbre in liste	recherche si nbre est dans la liste
a and b	évalue si a et b sont réalisées en même temps
a or b	évalue si a ou b sont réalisées
not a	évalue si non a est réalisée

Opérations

a//b	quotient de la division entière de a par b
a%b	reste de la division entière de a par b
a**b	a exposant b
a += 1	incrémenter (augmenter) la variable de 1

Saisie et affichage

nom = input("Entrez votre nom :")	nom est de type str
age = int(input("Entrez votre âge :"))	age est de type int
print("Coucou", nom)	affiche la variable nom et la chaine entre guillemets
print("a", "b", sep = ";")	les deux lettres sont séparés par ;
print("a", "b", end = "")	évite le retour à la ligne après l'affichage

Modules ou Bibliothèques

from math import*	on importe toutes les méthodes du module math
from math import sqrt	on importe seulement la méthode sqrt (racine carré)
import math	on importe tout le module et
	on écrit math.sqrt() pour utiliser la méthode sqrt()
dans le module random :	
randint(1,6)	donne un entier au hasard entre 1 et 6
choice([1,2,3,4,5])	donne un élément de la liste au hasard

Condition

```
if condition1 :      # si la condition 1 est vraie
    traitement1
elif condition2 :    # si la condition 2 est vraie
    traitement2
else :               # si aucune des deux conditions n'est vraie
    traitement3
```

Boucles while

while condition :	tant que la condition est vraie
instructions	on répète les instructions

Boucles for

for i in range(n) :	i prends les valeurs de 0 à n-1
instructions	on répète n fois
range(2,n)	i prends les valeurs de 2 à n-1
range(1,n,2)	i prends les valeurs de 1 à n-1, avec un pas de 2

Objet : Chaine - Tuple - Liste

len(objet)	longueur de l'objet
objet[i]	donne l'élément d'indice i
objet[p :q]	sélectionne de l'indice p à l'indice q-1
objet[p :]	sélectionne de l'indice p jusqu'à la fin
objet[:q]	sélectionne du début à l'indice q-1
objet[-1]	donne le dernier élément
objet1 + objet2	concaténation

Parcours d'une Chaine - Tuple - Liste

```
for element in objet :          parcours du contenu de l'objet
    print(element)
ou
for indice in len(objet) :      parcours à l'aide des indices
    print(objet[i])
```

Listes (Tableaux)

liste = [ ]	création d'une liste vide
liste.append(element)	ajout de l'élément en fin de liste
liste.insert(i,element)	insérer l'élément à l'indice i
del liste[indice]	supprimer le terme dont on donne l'indice
liste.remove(element)	supprime la première occurrence de l'élément

Dictionnaires

dico = {}	création d'un dictionnaire vide
dico[clé]	affiche la valeur correspondant à la clé donnée
dico[clé] = valeur	ajoute la clé et la valeur dans le dictionnaire
dico.keys()	donne la liste des clés du dictionnaire
dico.values()	donne la liste des valeurs du dictionnaire
dico.items()	donne la liste des clés et de leurs valeurs sous forme de tuples

Parcours d'un dictionnaire

```
for cle in dico.keys() :                on parcourt les clés du dictionnaire
    print(cle)
```

```
for valeur in dico.values() :           on parcourt les valeurs du dictionnaire
    print(valeur)
```

```
for cle,valeur in dico.items() :        on parcourt les tuples (clés,valeurs) du dictionnaire
    print(cle,valeur)
```

Création en compréhension de listes et dictionnaires

liste =[i for i in range(10)]	création de la liste des entiers de 0 à 9
liste =[i for i in range(10) if i%2 ==0]	si le nombre est pair
dico = {x : 2*x for x in range(5)}	création d'un dictionnaire

Fonctions

```
def nom_fonction(parametre1, parametre2) :   on donne des paramètres nécessaires au traitement
    """la fonction sert à..."""              docstring (description de la fonction)
    traitement
    return resultat                          le résultat est retourné en tant que variable
```

nom_fonction(n1,n2)	on appelle la fonction en précisant les paramètres
assert nom_fonction(n1,n2) == resultat	on teste si le résultat est bien celui attendu avec les paramètres n1 et n2 donnés
help(nom_fonction)	affiche la docstring de la fonction