Harvard CS50W
Introduction à Python

Franz Girardin

May 8^{th} 2023

TABLE DES MATIÈRES

CHAPITRE 1

CRÉATION DE FICHIER PYTHON

Exemple.

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

La fonction `print()` prend un l'argument entre parenthèse et *l'affiche en ligne de commande*. On enregistre un fichier avec l'extension `.py`. Pour lancer un programma Python, on navigue verse le répertoire contenant le fichier et on tape `python nomFichier.py`. Lorsqu'on lance cette commande, un interprète lie le fichier ligne par ligne et l'exécute en temps réel.

CHAPITRE 2

VARIABLES ET TYPES

Syntaxe.

La création de variable suit la syntaxe suivante : `nomVariable = valeur`

Exemple. Création de varibles

```
a = 28
b = 1.5
c = "Hello!"
d = True
e = None
```

TABLE 2.1 – Principaux types primtifs Python

Type	Description
`int`	*Un entier*
`float`	*Un nombre décimal*
`chr`	*A simple caractère*
`str`	*Une chaîne de caractère*
`bool`	*Une valeur qui est soit* `vrai` *ou* `fausse`
`chr`	*A simple caractère*

Exemple. Entrée d'utilisateur

```
    # Ce programme prends comme entree le nom de l'utilisateur et l'imprime en ligne de commande
    name = input("Name: ")
    print("Hello, " + name)
    # On utilise la fonction input() pour enregistre l'entree
```

2.1 Formater une chaîne de caractère

Syntaxe. Générer des *fString*

On utilise la syntaxe `f"Expression {variable}"` pour concaténer une chaîne de caractère existante à la chaîne correspondante d'une variable.

Exemple. Concaténation par *fString*

```
    # Methode f string.
    print(f"Hello, {input("Name: ")}")

    # Methode classique
    print("Hello, " + name)
```

2.2 Conditions

Syntaxe. Assertions conditionnelles

Elles sont introduites par les mots clés `if` `elif` `else` suivit de deux points `:`. L'indentation indique que *l'expression suivante sera évaluée à* `True` ou `False`

Exemple. Assertions conditionnelles

```
# Si on n'avait pas convertit l'entree en entier, l'interpretteur aurait lance une exception.
num = int(input("Number: "))
if num > 0:
    print("Number is positive")
elif num < 0:
    print("Number is negative")
else:
```

```
    print("Number is 0")

```

2.3 Séquences

Définition Séquence Mutable

On dit qu'une séquence est mutable lorsqu'il est possible de changer les éléments individuels de cette *séquence* après sa création.

Définition Séquence ordonnée

Séquence dans laquelle l'ordre des éléments est important.

Note :

Les chaîne de caractères sont **ordonnées** et **immuables**

Exemple. Accéder à un caractère

```
        # On peut acceder a un caractere en specifiant la position de celui-ci par rapport la variable
        name = "Harry"
        print(name[0]) # Renvoit "H"
        print(name[1]) # Renvoit "a"


```

2.4 Listes

Définition Liste Python

Structure de données qui nous permet *d'enregistrer n'importe quel type*. Une liste est contenu par `[...]`.

Exemple. Création de liste

```

names = ["Harry", "Ron", "Hermione"]
# Imprime la liste entiere
print(names)
# Imprime le second element de la liste
print(names[1])
# Ajoute un nouveau nom a la liste
names.append("Draco")
# Organise la liste en ordre alphabetique
names.sort()
# Imprime la nouvelle liste
print(names)
```

Note :

Les listes sont **ordonnées** et **mutables**.

2.5 Tuples

Définition Tuples

Structure de données généralement utilisée pour enregistrer *deux ou trois types de valeurs*—des coordonnées par exemple.

Exemple. Création d'un tuple

```
# On l'utilise ici pour creer un systeme de coordonees
point = (12.5, 10.6)
```

Note :

Les tuples sont **ordonnés** et **immuables**.

2.6 Ensembles

Définition Ensemble

Un ensemble est un types où une valeur *ne peut être enregistré plus d'une fois.*

Exemple. Manipulation des ensembles

```
# Cree un ensemble vide
s = set()

# Ajoute des elements dans l'ensemble via la fonction add().
s.add(1)
s.add(2)
s.add(3)
s.add(4)
s.add(3)
s.add(1)

# Retire la valeur 2 de l'ensemble.
s.remove(2)

# Imprime l'ensemble
print(s)

# Determine et affiche la taille de l'ensemble
print(f"The set has {len(s)} elements.")

""" Ceci est un commentaire de plusieurs lignes en Python
Output:
{1, 3, 4}
The set has 3 elements.
"""
```

2.7 Dictionnaires

Définition Dictionnaire

Ensemble de `clé:valeur` ; pour chaque clée, il y a une valeur correspondante.

Exemple. Création de dictionnaire

```python
# Definit un dictionnaire
houses = {"Harry": "Gryffindor", "Draco": "Slytherin"}

# Imprime la maison a laquellle Harry apparient.
print(houses["Harry"])

# Ajoute une valeur au dictionnaire.
houses["Hermione"] = "Gryffindor"

# Imprime la maison a laquelle Hermione appartient.
print(houses["Hermione"])

""" Output:
Gryffindor
Gryffindor
"""
```

Note :

Les dictionnaires `non ordonnées` et `mutables`

2.8 Boucles

Définition

Utilisé pour itérer sur une séquence d'élément en appliquant un bloc d'instructions.

Exemple. Utilisation de boucle for

```python
# Imprime les nombres de 0 a 5.
for i in [0, 1, 2, 3, 4, 5]:
    print(i)

""" Output:
0
1
2
3
4
5
"""
# Alternative en utilisant range() pour alleger l'ecriture
for i in range(6):
    print(i)

""" Output:
0
1
```

```
2
3
4
5
"""
# Cree une liste
names = ["Harry", "Ron", "Hermione"]

# Imprime chaque nom de la liste
for name in names:
    print(name)

""" Output:
Harry
Ron
Hermione
"""

Iterer sur une chaine de caractere :
name = "Harry"
for char in name:
    print(char)

""" Output:
H
a
r
r
y
"""
```

2.9 Fonctions

Exemple. Création de fonction

```
# Cette fonction prend comme argument un nombre et retourne son carre
def square(x):
    return x * x

for i in range(10):
    print(f"The square of {i} is {square(i)}")

""" Output:
The square of 0 is 0
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
The square of 6 is 36
The square of 7 is 49
The square of 8 is 64
The square of 9 is 81
"""
```

2.10 Modules

Note :

Il est possible d'importer des fonctions contenu dans un fichier pour les utiliser dans un autre

Exemple. Importer une fonction

```python
# Contenu du fichier functions.py
def square(x):
    return x * x
# ================================================================================
# Contenu fu fichier square.py

from functions import square # Importe la fonction square du fichier functions.py
for i in range(10):
    print(f"The square of {i} is {square(i)}")

# ================================================================================
# Alternative

import functions # On importe le module au complet
for i in range(10):
    print(f"The square of {i} is {functions.square(i)}")
```

2.11 Programmation Orientée Objet

Définition

Il s'agit d'un paradigme de programmation qui met de l'avant les objets qui peuvent *enregistrer de l'information et effectuer des actions.*

Définition Classes Python

Ce sont des types définit par le programmeur. C'est essentiellement le patron pour un nouveau type d'objet qui peut enregstrer de l'information et effectuer des actions.

Exemple. Créer une classe

```python
class Point():
    # A method defining how to create a point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

# Utilisation de la classe Point()
p = Point(2, 8)
print(p.x)
print(p.y)

""" Output:
2
8
"""
# ================================================================================
class Flight():
    # Methode qui cree un nouveau vol avec un capacite donnee
```

```
    def __init__(self, capacity):
        self.capacity = capacity
        self.passengers = []

    # Method qui ajoute un passager au vol
    def add_passenger(self, name):
        if not self.open_seats():
            return False
        self.passengers.append(name)
        return True

    # Methode qui retourne le nombre de places disponibles
    def open_seats(self):
        return self.capacity - len(self.passengers)
# ================================================================================
# Utilisation de la classe Flight()

# Cree un nouveau vol avec capacite de 3 passagers
flight = Flight(3)

# Cree une liste de personnes
people = ["Harry", "Ron", "Hermione", "Ginny"]

# Tente d'ajoute chaque personne de la liste au vol
for person in people:
    if flight.add_passenger(person):
        print(f"Added {person} to flight successfully")
    else:
        print(f"No available seats for {person}")

""" Output:
Added Harry to flight successfully
Added Ron to flight successfully
Added Hermione to flight successfully
No available seats for Ginny
"""
```

2.12 Programmation fonctionnelle

Définition Programmation fonctionnelle

Paradigme de programmation où on traite chaque fonction comme n'importe quelle autre variable.

Exemple.

```
# On peut simplifier l'ecriture d'une fonction courte grace a lambda
square = lambda x: x * x
# ================================================================================
# Organiser une liste de dictionnaire
people = [
{"name": "Harry", "house": "Gryffindor"},
{"name": "Cho", "house": "Ravenclaw"},
{"name": "Draco", "house": "Slytherin"}
]

def f(person):
```

```python
    return person["name"]

people.sort(key=f)

print(people)

""" Output:
[{'name': 'Cho', 'house': 'Ravenclaw'}, {'name': 'Draco', 'house': 'Slytherin'}, {'name': 'Harry
    ', 'house': 'Gryffindor'}]
"""
# ================================================================================
# Methode avec fonction lamda

people = [
    {"name": "Harry", "house": "Gryffindor"},
    {"name": "Cho", "house": "Ravenclaw"},
    {"name": "Draco", "house": "Slytherin"}
]

people.sort(key=lambda person: person["name"])

print(people)

""" Output:
[{'name': 'Cho', 'house': 'Ravenclaw'}, {'name': 'Draco', 'house': 'Slytherin'}, {'name': 'Harry
    ', 'house': 'Gryffindor'}]
"""
```

2.13 Exceptions

Syntaxe. Gestion des exceptions

Il arrive qu'on effectue des opérations qui lèvent une exception. On peut gérer ces instances avec un les commandes `try` et `excpet`

Exemple. Division par zéro

```python
# Ce programme risque de lever une exception si le diviseur (y) est zero
x = int(input("x: "))
y = int(input("y: "))

result = x / y
# ================================================================================

# L'alternative est un programme incorporant try et except.
import sys

x = int(input("x: "))
y = int(input("y: "))

try:
    result = x / y
except ZeroDivisionError:
    print("Error: Cannot divide by 0.")
    # Exit the program
    sys.exit(1)
```

```
20
21 print(f"{x} / {y} = {result}")
22
23
24
25 print(f"{x} / {y} = {result}")
```