Formation Python

Arcadius SOGLO

Introduction

Programme informatique

- Ensemble d'instructions exécutés par l'ordinateur
- Les instructions doivent respectés une grammaire et des règles propre à chaque langage

Code source

- il s'agit du texte écrit par le programmeur

Compilateur

Interpreteur

Langage compilé

- C
- C++
- Pascal
- OCaml

Langage interpreté

- Python
- JAVA

Langage haut niveau

- Comporte des instructions plus abstraites, proche des langues naturelles (anglais)
- Exemple : Python, JAVA, C, ...

Langage bas niveau

- Constitué d'instructions proche de la machine
- Exemple : Assembleur

La démarche du programmeur

- L’activité essentielle d’un programmeur consiste à résoudre des problèmes
- Les erreurs de programmation s’appellent des **bugs** ou en français **bogues**
- L’ensemble des techniques que l’on met en oeuvre pour détecter et corriger les erreurs s’appelle **debug** ou ou en français **débogage**

Types d'erreurs

- **Erreurs de syntaxe**

Le terme syntaxe se réfère aux règles que les auteurs du langage ont établies pour la structure du programme.

- **Erreurs sémantiques**

Il s'agit des fautes de logique

- **Erreurs à l’exécution (Run-time error)**

Ces erreurs sont également appelées des exceptions.

Caractéristiques du langage python

- Développé en 1989
- Langage portable (Unix, MacOS, Windows)
- Langage orienté objet
- Typage dynamique
- Langage Interprété assisté d'un pré-compilateur
- Extensible et blibliotheque standard riche

Versions

Version	Année de sortie	Commentaires
2.0	Octobre 2000	Fin de support en 2020
...	...	
3.0	Décembre 2008	
...	...	
3.7	Juin 2018	
3.8	Octobre 2019	

Les différentes utilisations du langage

- Automatisation (paramiko, netmiko)
- Création de site web (Django, Flask)
- Interface graphique (Tkinter, PyQT)
- Intelligence artificielle (scikit-learn, tensorflow)
- Science des données et Big Data(pandas, numpy)
- etc.

Installation Python

- http://www.python.org
- https://docs.python.org/3.7/

Les interpreteurs Python

- CPython (C, par défaut)
- Jython (JAVA)
- IronPython (.NET)
- PyPy (Python)

Vérification Installation Python

- version

```
C:\> python --version
Python 3.7.1
```

- console python

```
C:\> python
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:05:16) [MSC v.1915 32 bit (Intel
Type "help", "copyright", "credits" or "license" for more information.
>>>1+1
2
```

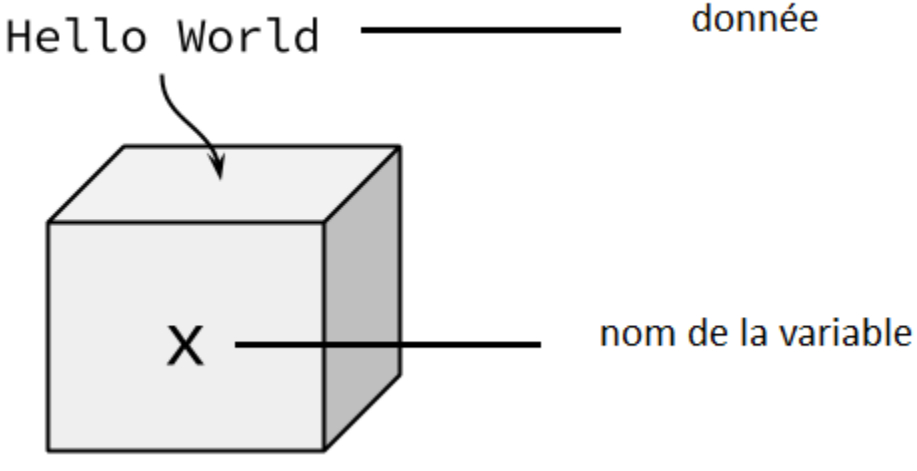
Installation IDE

- idle
- notepad++
- **vscode**
- pycharm
- atom
- eclipse

Les concepts du langage

- Variables
- Opérateurs
- Controle flux d'exécution
- Structures de données
- Fonctions
- POO

Variables


Hello World
donnée
X
nom de la variable

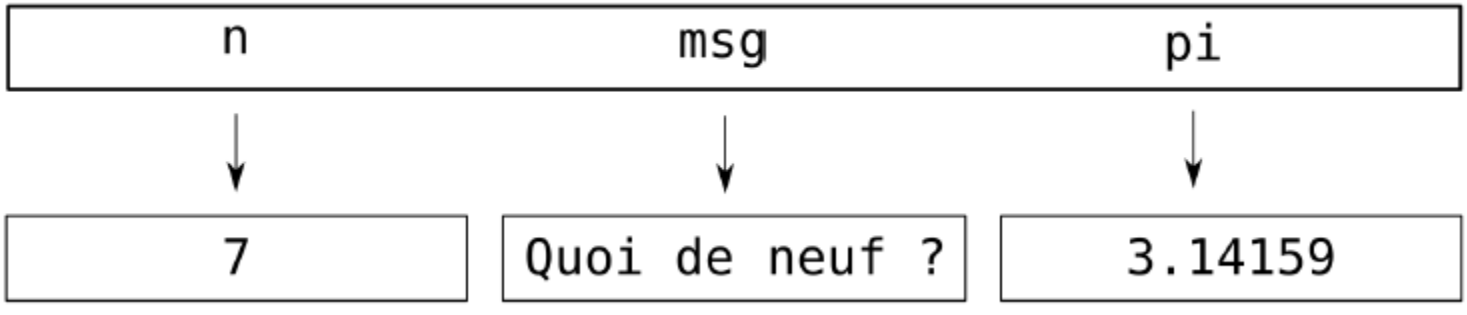
Règles noms de variables

- (a → z , A → Z) et de chiffres (0 → 9), qui doit toujours commencer par une lettre
- Seules les lettres ordinaires sont autorisées (à l'exception du caractère _)
- Joseph != joseph != JOSEPH
- PEP8 : **[a-z*][a-z0-9*]{2,30}$**

Les mots réservés

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

Déclaration de variable



- définir la variable n et lui donner la valeur 7

```
n = 7
```

- affecter la valeur "Quoi de neuf ?" à msg

```
msg = "Quoi de neuf ?"
```

- assigner sa valeur à la variable pi

```
pi = 3.14159
```

Afficher la valeur d'une variable

- console

```
>>> n
7
>>> msg
'Quoi de neuf ?'
>>> pi
3.14159
```

- print()

```
>>> print(msg)
Quoi de neuf ?
>>> print(n)
7
```

Typage dynamique

```
>>> var_integer = 1
>>> var_float = 3.14
>>> var_string = "Hello World"
>>> var_boolean = True
```

Assigner une valeur à plusieurs variables

```
>>> x = y = 7
>>> x
7
>>> y
7
```

Affectations parallèles

```
>>> a, b = 4, 8.33
>>> a
4
>>> b
8.33
```

Opérateurs

Opérateurs

- Parenthèse: **()**
- Exposant: ******
- Multiplication: *
- Division: **/**
- Division entière: **//**
- Adition: **+**
- Soustraction: **-**
- Modulo: **%**
- Affectation: **=**
- Comparaison: **==** , **<** , **>** , **!=**, **<=** , **>=**

Controle flux d'exécution

- Sélection ou exécution conditionnelle
- Instructions répétitives

Sélection ou exécution conditionnelle

```
a = 20
if (a > 100):
    print("a dépasse la centaine")
else:
    print("a ne dépasse pas cent")
```

Répétitions en boucle : while

```
a, b, c = 1, 1, 1
while c < 11:
    print(b, end =" ")
    a, b, c = b, a+b, c+1
```

Variables	a	b	c
Valeurs initiales	1	1	1
Valeurs prises successivement, au cours des itérations	1 2 3 5 ...	2 3 5 8 ...	2 3 4 5 ...
Expression de remplacement	b	a+b	c+1

Répétitions en boucle : for … in …

```
a, b = 1, 1
for c in range(1, 12):
    print(b, end =" ")
    a, b = b, a+b
```

Variables	a	b	c
Valeurs initiales	1	1	1
Valeurs prises successivement, au cours des itérations	1 2 3 5 …	2 3 5 8 …	2 3 4 5 …
Expression de remplacement	b	a+b	c+1

Context Manager with

```
with open("/etc/passwd") as fp:
    content = fp.read()
    print(content)
```

Gestion des exceptions try / except / finally / else

```
try:
    1/0
except Exception as exc :
    print ("erreur = {}".format(exc))
else:
    print ("else")
finally:
    print ("finally")
```

Structures de données

Rappel type primitif

- type integer

```
var_integer = 1
```

- type float

```
var_float = 3.14
```

- type string

```
var_string = "Chaine de caractère"
```

Liste - Tableau - Array

- Déclaration liste

```
jour = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi']
```

- Taille d'une liste

```
len(jour)
```

- Accéder à des éléments de la liste

```
print(jour[0]) # premier élément
print(jour[0:3]) # trois premiers éléments
print(jour[-1]) # dernier élément
print(jour[-4:]) # quatre derniers éléments
```

- Ajouter élément à la liste

```
jour.append('samedi')
```

- Supprimer élément de la liste

```
del(jour[-1])
jour.remove("samedi")
```

Les tuples

- Les tuples sont non immutable (non modifiiable)
- Les fonctions append, del, et remove ne s'appliquent pas aux tuples
- Déclaration tuple

```
jour = ('lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi')
```

- Taille tuple

```
len(jour)
```

- Accéder à des éléments

```
print(jour[0]) # premier élément
print(jour[0:3]) # trois premiers éléments
print(jour[-1]) # dernier élément
print(jour[-4:]) # quatre derniers éléments
```

Les dictionnaires

- Déclaration d'un dictionnaire

```
dico = {
    'computer': 'ordinateur',
    'keyboard': 'clavier',
    'mouse': 'souris'
}
```

- Méthodes spécifiques

```
print(dico.keys())

print(dico.values())

print(dico.items())
```

- Taille dictionnaire

```
len(dico)
```

Les dictionnaires

- Accéder à des éléments du dictionnaire

```
print(dico['mouse']) # afficher un élément spécifique

# afficher tous les éléments
for key,value in dico.items():
    print("%s : %s" %(key : value))
```

- Ajouter un élément au dico

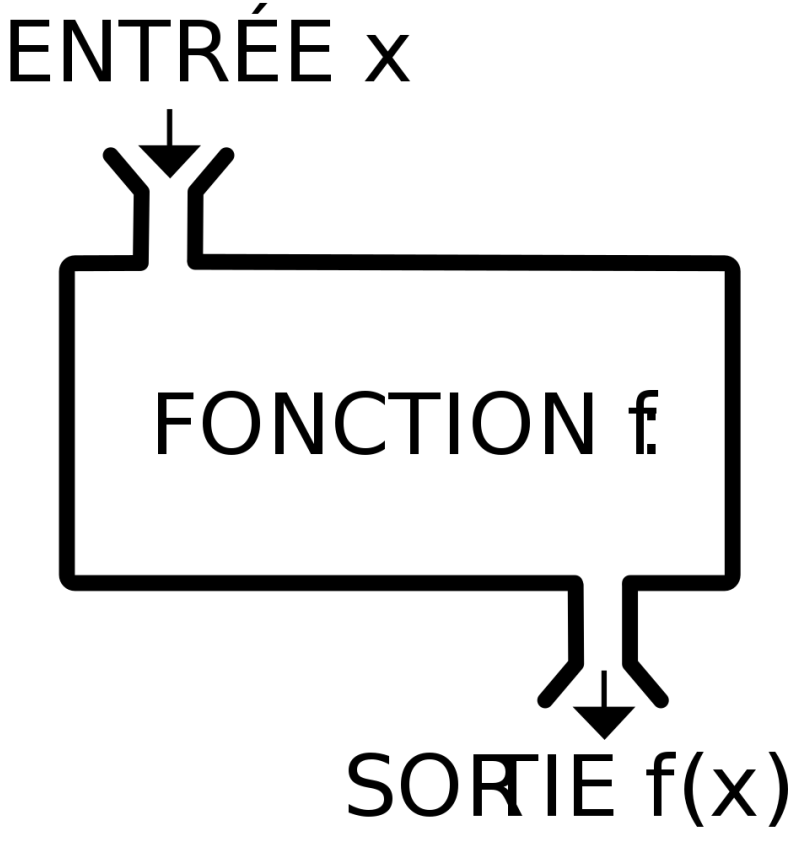
```
dico["pineapple"] = "ananas"
dico["apple"] = "pomme"
```

- Supprimer un élément du dico

```
del(dico["pomme"])
```

Les fonctions

- Une procédure ne renvoit rien



- Une fonction doit retourner une valeur

Les fonctions prédéfinies

		Built-in Functions		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

Définir une fonction

```
# Volume d'un sphere

def cube(nombre):
    """docstring : Ceci est un commentaire
    """
    return nombre**3

def volumeSphere(rayon):
    return 4 * 3.1416 * cube(rayon) / 3
```

Variables locales, variables globales

- Les variables définies à l'extérieur d'une fonction sont des variables globales
- Les variables définies à l'intérieur d'une fonction sont des variables locales
- Les variables définies localement ont la priorité en cas de conflit(meme nom de variable)

```
>>> def mask():
        p = 20
        print(p, q)

>>> p, q = 15, 38

>>> mask()
20 38

>>> print(p, q)
15 38
```

Variables locales, variables globales

global

```
>>> def plus_un():
        global a
        a = a+1
        print(a)

>>> a = 15
>>> plus_un()
16

>>> plus_un()
17
>>>
```

Fonctions de la librairie standard

https://docs.python.org/3/library/

- Numeric and Mathematical Modules
  - `numbers` — Numeric abstract base classes
  - `math` — Mathematical functions
  - `cmath` — Mathematical functions for complex numbers
  - `decimal` — Decimal fixed point and floating point arithmetic
  - `fractions` — Rational numbers
  - `random` — Generate pseudo-random numbers
  - `statistics` — Mathematical statistics functions

```
from math import sqrt
print(sqrt(4))

import random
print(random.random())
```

Fonctions de la librairie communautaire

Exemple : numpy

https://docs.scipy.org/doc/numpy/reference/index.html

- Installation de la librairie

```
C:\> pip install numpy
C:\> pip install numpy --index-url https://artifactory-iva.si.francetelecom.fr/ar
```

- Utilisation des fonctions de la librairie

```
from numpy.random import rand
values = rand(10)
print(values)
```

Programmation orienté objet

Principes

L'approche objet permet de modéliser un systeme par rapport aux entités du monde réel

- applications complexes (réutilisabilité, modularité)
- applications évolutives (évolutibilité)

Principes

Classe

- définit une réprésentation abstraite d'une entité
- propriétés communes à un ensemble d’objets
- modéle à partir duquel les objets peuvent être crées

Exemple

- Personne
- Voiture

Principes

Objet

- définit une réprésentation physique d'une entité
- c'est une **instance** d'une classe
- Objet = identifiant + état + comportement

Exemple

- objet1 : Personne, ("Martin", "12, pl. Leclerc, Lannion", 31/12/2003)
- objet2 : Voiture, ("Laguna", "Coupé", "Blanche", 7, 2013)

Principes

Encapsulation

- l'implémentation est cachée
- masque la complexité à l'utilisateur
- l'objet n'est accessible que par son interface

Principes

Héritage

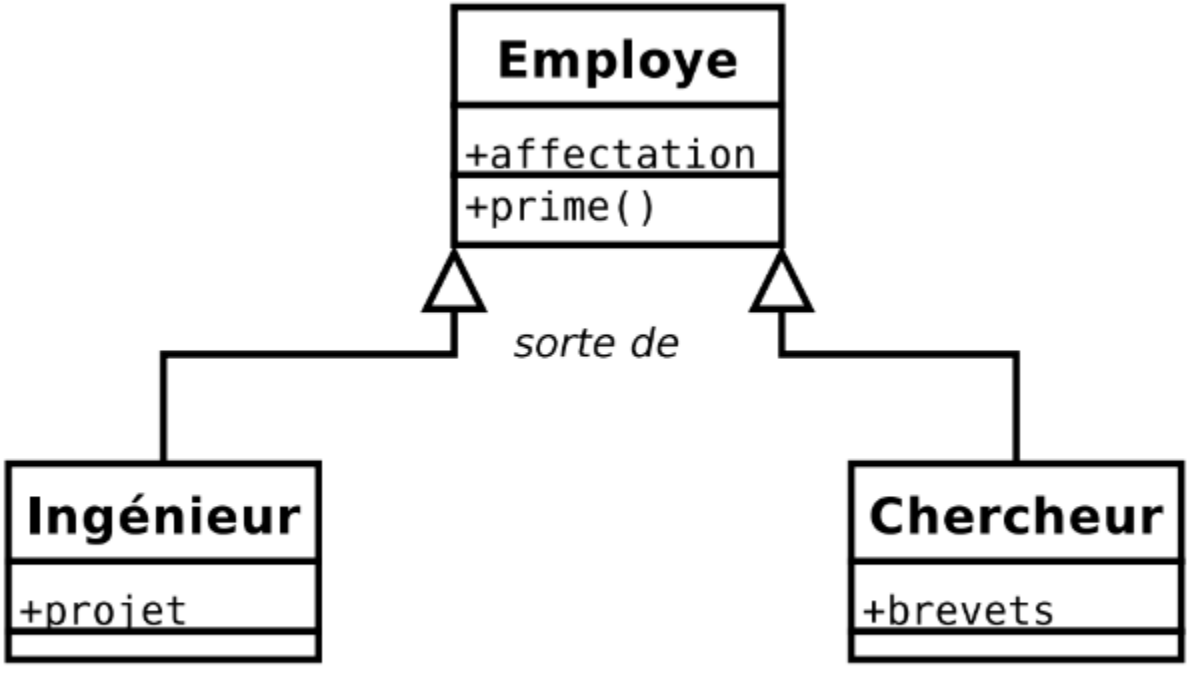
- Possibilité pour une classe de dériver d'une ou de plusieurs classes
- Héritage simple ou multiple

Principes

Polymorphisme

- Un objet peut prendre plusieurs formes
- permet d'attribuer des comportements différents à des objets en fonction du contexte

Principes



Modélisation UML

- Constructeur (initialisation de l'objet)
- Attributs et Méthodes (public, private, protected)

Définition d'une classe

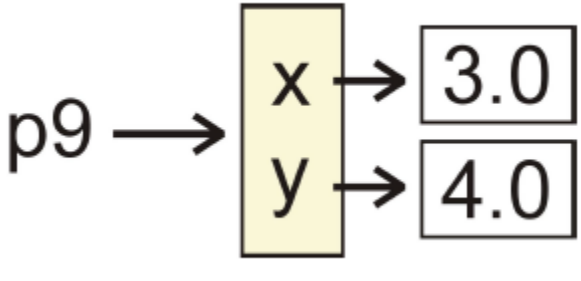
```
class Point(object):
    "Définition d'un point géométrique"
    pass
```

Objet : instance d'une classe

```
p9 = Point()

print(p9)

print(p9.__doc__)
```

Attributs (variables) d'instance

```
>>> p9.x = 3.0

>>> p9.y = 4.0
```



Public/Private

Private instance variables that cannot be accessed except from inside an object don't exist in Python

docs.python.org

- Simple underscore (_spam) pour indiquer que la variable ou la méthode est **"private"**
- Possibilité de masquer la variable ou la méthode dans l'interpreteur avec le double underscore (__spam)

Tutoriels Exemple

La méthode constructeur

```
class Time(object):
    "Encore une nouvelle classe temporelle"
    def __init__(self, hh=12, mm=0, ss=0):
        self.heure = hh
        self.minute = mm
        self.seconde = ss
    def affiche_heure(self):
        print("{0}:{1}:{2}".format(self.heure, self.minute, self.seconde))


recreation = Time(10, 15, 18)
recreation.affiche_heure()
```

Héritage

```
class Mammifere(object):
    caract1 = "il allaite ses petits ;"

class Carnivore(Mammifere):
    caract2 = "il se nourrit de la chair de ses proies ;"

class Chien(Carnivore):
    caract3 = "son cri s'appelle aboiement ;"


mirza = Chien()
print(mirza.caract1, mirza.caract2, mirza.caract3)
```

Héritage multiple

- Diamond Problem

```
class UnObjet:
    nom = None
```

```
class Vehicule(UnObjet):
    nombre_roue = 4
```

```
class Voiture(Vehicule):
    nombre_porte = 4
```

```
class VoitureSport(Voiture):
    couleur = 'rouge'
```

```
class Renault(Voiture):
    nom = 'renault'
```

```
class Ferrari(Voiture, VoitureSport):
    nom = 'ferrari'
```

QUIZZ

THANKS !