[supDeCode]
®

Avril 2021

contact@supdecode.fr

Objectifs

- Maîtriser le langage en programmation procédurale et POO.

Pré-requis

- Aucun

Contenu (1/3)

- Historique
- Installation
- Syntaxe de base
- Types
- Littéraux
- Casts
- Variables
- Opérateurs d'affectation
- Immutabilité

Contenu (2/3)

- Opérateurs logiques
- Opérateurs de comparaison
- Tests conditionnels
- Séquences : tuples, listes, ranges
- Dictionnaires
- Jeux
- Structures itératives
- Fonctions
- Portée des variables
- Modules

Contenu (3/3)

- Classes
- Membres d'instance et membres statiques
- Propriétés dynamiques
- Constructeurs
- Héritage simple
- Duck-typing
- Redéfinitions
- Héritage multiple
- Méthodes magiques
- Exceptions

Outils

- Interpréteur Python et ses accessoires.
  //www.python.org
- VS Code avec l'extension Python de Microsoft.
  //code.visualstudio.com

Historique

- Créé par Guido van Rossum (NL) en 1991.
  Clin d'oeil aux Monty Python...
- Version 2 en 2000.
- Version 3 en 2008 incompatible avec la version 2.
- Actuellement en version 3.9.

Caractéristiques

- Open-source et gratuit.
- Interprété donc sans typage statique.
- Utilisable en POO et/ou en procédural.
- Désallocation mémoire via Garbage Collector.

Domaines d'utilisation

- Initiation à la programmation car proche du pseudo-code.
- Traitements mathématiques (Bibliothèque Sage).
- Langage de script dans Autodesk : Autocad, Revit, Maya, 3DS Max....
- Interfaces, "glue" entre applicatifs.
- Applications web avec le framework Django.
- Applications multi-plateformes avec le framework Qt.

Documentation

- Site officiel.
  //www.python.org
- Documentation officielle.
  //docs.python.org
- Association francophone pour la promotion de Python.
  //www.afpy.org

Installation sous Windows

- Binaire disponible sur le site officiel.
- Ajoute le chemin de l'exécutable dans le PATH.
- Fonctionne immédiatement via l'invite de commandes de Windows.
- `> python`
  `> py`
- VS Code fonctionne dès que l'interpréteur est défini.

Premières lignes de code

```
a = "Toto"
b = "Lulu"
print(f"{a} aime {b}") #Toto aime Lulu
```

Premier programme

- Créez le fichier **`hello.py`** contenant la ligne :

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

- Lancez son exécution :

```
> py hello.py

#Hello !
```

Syntaxe

- Python est intégralement sensible à la casse (mots-clés, identifiants...).
- La fin de ligne suffit à terminer une ligne de code.
- Les expressions entre `()`, `[]` ou `{}` peuvent s'étendre sur plusieurs lignes.
- Le ; permet de séparer plusieurs instructions sur une même ligne.
- Indentation obligatoire (espaces recommandés).
- Commentaires d'une ligne avec #.
  Pas de commentaires multilignes.

Afficher

- Fonction top-level **`print()`**.
  Syntaxe : **`print(`*`exp`*`, sep = ' ', end = '\n')`**
- Exemples :

```
print('toto') #toto
print('toto', 'lulu') #toto lulu
print('toto', 'lulu', sep = ':', end = ';')
#toto:lulu;
```

- Arguments nommés (*keyword arguments*) : voir plus loin.
- La méthode **`printf()`** est obsolète. Voir plus loin **`format()`**.
- Saut de ligne : **`\n`**
  Tabulation : **`\t`**

Saisir

- Fonction *builtin* **`input()`**.
  Syntaxe : **`input(`*`prompt`*`)`**
- Affiche le prompt si présent, arrête l'exécution puis retourne la saisie.
- Saisie toujours considérée comme une chaîne.
  Retour à la ligne non capturé.
- Exemples :

```
prenom = input("Prénom ? ")

age = int(input("Age ? "))
```

Terminer

- Fonction top-level **`exit()`**.
  Syntaxe : **`exit(`*`message`*`)`**
- Affiche le message puis arrête l'exécution.

Fonctions *builtins*

- Fonctions regroupées dans le module **`builtins`** (voir plus loin).
- Peuvent être appelées directement sans référence à un module.
- Exemples déjà rencontrés :
  - **`print()`**
  - **`input()`**
  - **`exit()`**

Types *builtins*

- Types (classes) regroupés dans le module **`builtins`** (voir plus loin).
- Peuvent être utilisés directement sans référence à un module.
- Quelques exemples :
  - **`int`**
  - **`float`**
  - **`complex`**
  - **`bool`**
  - **`str`**
- La fonction *builtins* **`type()`** retourne le type d'une donnée.
  Exemple :

```
type(123) #<class 'int'>
```

Littéraux entiers

- Instances de la classe **`int`**.
- **Longueur illimitée.**
- Exemples :
  - **`123`** (décimal)
  - **`0b1111011`** (binaire)
  - **`0o173`** (octal)
  - **`0x7B`** (hexadécimal)
  - **`(123).bit_length() #7`**
    Ce qui prouve qu'un littéral entier est une instance !

Littéraux réels

- Instances de la classe **`float`**.
- Exemples :
  - **`123.0`**
  - **`.123`**
  - **`1.23e2`**
  - **`(123.0).is_integer() #True`**

Littéraux complexes

- Instances de la classe **`complex`**.
- Exemples :
  - `3j` (partie réelle nulle)
  - `4+3j`
  - `1j ** 2 #(-1+0j)`

Littéraux booléens

- Deux mot-clés :
  - **`True`** (vaut 1)
  - **`False`** (vaut 0)
- Exemples :

```
True #True
True + 1 #2
```

Evaluation booléenne

- Les valeurs évaluées à faux sont :
  - `None` (constante "rien", comparable et affectable)
  - `False`
  - `0` (ou `0.0`)
  - `''` (ou `""`)
  - `()` (tuplet vide, voir plus loin)
  - `[]` (liste vide, voir plus loin)
  - `{}` (jeu vide, voir plus loin)
- Toutes les autres valeurs sont évaluées comme vraies.

Conversions entre types numériques

Conversions implicites

- Exemples :
  - `2 + 3.0 #5.0`
  - `10 * False + 100 * True #100`

Conversions explicites

- Exemples avec les fonctions *builtins* :
  - `int(5.9) # 5`
  - `float(5) # 5.0`
  - `bool(5)  # True`
  - `bool(0)  # False`
  - `bool(-1) # True`

Opérateurs arithmétiques

`+`	Addition, ex : `5 + 2 #7` Opérateur unaire, ex : `+3 #3`
`-`	Soustraction, ex : `5 - 2 #3` Opérateur unaire, ex : `-3 #-3`
`*`	Multiplication, ex : `5 * 2 #10`
`/`	Division réelle, ex : `5 / 2 #2.5`
`//`	Division entière, ex : `5 // 2 #2`
`%`	Modulo, ex : `5 % 2 #1`
`**`	Puissance, ex : `5 ** 2 #25`

Fonctions *builtins* arithmétiques

- Quelques exemples :

`abs(x)`	Valeur absolue, ex : `abs(-3) #3`
`max(...)`	Maximum d'une séquence. Ex : `max(3, 5, 2) #5`
`min(...)`	Minimum d'une séquence. Ex : `min(3, 5, 2) #2`
`round(x, n)`	Arrondi (alias de math.round). Ex : `round(3.1416, 3) #3.142`

Module `math`

- Quelques exemples de fonctions :

`math.pi`	Pi.
`math.e`	Base du logarithme népérien.
`math.ceil(x)`	Arrondi à l'entier supérieur. Ex : `math.ceil(2.1) #3`
`math.floor(x)`	Arrondi à l'entier inférieur. Ex : `math.floor(2.9) #2`
`math.trunc(x)`	Partie entière. Ex : `math.trunc(2.5) #2`
`math.log(x, b)`	Logarithme de `x` en base `b`. Ex : `math.log(8, 2) #3.0`

TD

facto123456

- A l'aide du module **`math`** :
  - Affichez la factorielle de **`123456`**.
  - Afficher le nombre de chiffres du résultat.
- Appréciez la rapidité de calcul de Python avec les nombres entiers !

Littéraux chaînes

- Instance de la classe **`str`**.
- Indifféremment entre simples ou doubles quotes.
- **Immutables.**
- Le caractère d'échappement est **`\`**.
- Concaténation implicite.
  Ex : **`print("toto" "aime" "lulu") #totoaimelulu`**
- Nombreuses méthodes disponibles.
  Exemples :

  `"toto".upper() #'TOTO'` (méthode d'instance)
  `str.upper("toto") #'TOTO'` (méthode statique)

Opérateurs sur chaînes

- Concaténation :

  `"Toto" + " aime " + "Lulu" #'Toto aime Lulu'`

- Répétition :

  `"la" * 3 #'lalala'`

- Fonction *builtin* `len()` :

  `len("3,99 €") #6` (Unicode)

Formatage des chaînes

- La fonction *builtin* **`format()`** ne permet de formater qu'une valeur.
  Exemple : **`format(12, '.2f') #12.00`**
- La méthode **`str.format()`** permet d'avantage.
  Exemple :

  `"{} oeufs = {:.2f} €".format(6, 3) #6 oeufs = 3.00 €`
- Documentation sur le formatage :
  //docs.python.org/3/library/string.html#formatspec

Les f-strings

- La meilleure façon d'afficher et de formater.
- Exemple :

```
ref = "A123"
prix = 5.2
print(f"L'article {ref} vaut {prix:.2f} €")
#L'article A123 vaut 5.20 €
```

Extraits de chaînes

- Extraction d'un caractère comme dans un tableau indicé.
  Exemples :

  `"Toto aime Lulu"[10] #'L'`

  `"Toto aime Lulu"[-4] #'L'` (indexation négative)

- Extraction d'une tranche (*slice*).
  Exemples :

  `"Toto aime Lulu"[5:8] #'aim'` (huitième exclu)

  `"Toto aime Lulu"[5:]  #'aime Lulu'`

  `"Toto aime Lulu"[:8]  #'Toto aim'`

  `"Toto aime Lulu"[:]   #'Toto aime Lulu'`

Conversions entre numériques et chaînes

- Aucune conversion implicite.
- Conversions explicites avec les fonctions *builtins*.
  Exemples :

```
str(5.9)          #'5.9'
float('5.9e3')    #5900.0
float('5')        #5.0
int('5')          #5
int('5.9')        Erreur, réel !
bool('toto')      #True
bool('')          #False
bool(0)           #False
bool('0')         #True
```

TD

factoStr

- Affichez le nombre de chiffres de la factorielle de **`123456`** en utilisant le module **`math`** uniquement pour le calcul de la factorielle elle-même.

Identifiants

- Pour tous les identifiants (variables, fonctions, classes...) :
  - Peuvent contenir a-z, A-Z, 0-9 et _.
  - Ne doivent pas commencer par un chiffre.
  - Longueur infinie.

Variables

- Pas de déclaration donc initialisation obligatoire.
- Typage dynamique.
- Exemple :

```
a = 3
a = "toto"
a = 3.5
print(a) #3.5
```

- Il n'existe pas de constantes en Python.

Opérateurs d'affectation

`=`	Affectation simple, ex : `a = 3` Affectation multiple (*parallel assignment*), ex : `a, b = 1, 2` Affectation chaînées, ex : `a = b = 3`
`+=`	Affectation et addition (ou concaténation) combinées.
`-=`	Affectation et soustraction combinées.
`*=`	Affectation et multiplication combinées.
`/=`	Affectation et division réelle combinées.
`//=`	Affectation et division entière combinées.
`%=`	Affectation et modulo combinés.
`**=`	Affectation et puissance combinées.

Types mutables et immutables

- Une variable d'un type immutable ne peut recevoir qu'une valeur unique dans son cycle de vie, à la façon d'une constante. Cependant, une nouvelle affectation crée automatiquement une nouvelle instance.
- Tous les types numériques ainsi que le type **`str`** sont immutables.
- Exemple (la fonction *builtin* **`id()`** retourne l'adresse mémoire) :

```
s = "toto"
print(id(s)) #44433792 (adresse mémoire de s)
s = "lulu"
print(id(s)) #36036424 (nouvelle adresse)
s += '!'
print(id(s)) #44522048 (nouvelle adresse)
s[1] = 'i'    Erreur, immutable !
```

Opérateurs de comparaison

`==`	Egalité simple.
`!=`	Non égalité simple.
`is`	Egalité stricte (même adresse mémoire).
`is not`	Non égalité stricte.
`<`	Strictement inférieur.
`<=`	Inférieur ou égal.
`>`	Strictement supérieur.
`>=`	Supérieur ou égal.

Opérateurs logiques

`not`	Négation.
`and`	Et.
`or`	Ou inclusif.

Tests conditionnels

- Syntaxe :

```
if exp1:
  ...code excécuté si exp1 vraie...
elif exp2:
  ...code excécuté si exp1 fausse mais exp2 vraie...
...
...
...
else:
  ...code exécuté si exp1 et exp2 fausses...
```

- **Attention à l'indentation obligatoire !**

Test conditionnel rapide

- Exemple :

```
valeur = 3
signe = 'positif' if valeur >= 0 else 'négatif'
print(signe) #positif
```

TD (1/3)

nir13

- Dans un premier temps, écrire un programme qui demande la saisie d'un numéro de sécurité sociale (NIR13 à 13 chiffres sans la clé) puis affiche la clé sur 2 digits.
- INFO
  La clé est le résultat de la soustraction à 97 du reste de la division du numéro par 97.
- Exemple :

```
NIR13 ? 2921059286404
Clé : 10
```

TD (2/3)

nir15

- Le programme demande maintenant la saisie du numéro de sécurité sociale complet avec la clé sous la forme :

  SAAMMDDNNNRRR-CC

- Il vérifie les points suivants :
  - S (sexe) est 1 ou 2.
  - MM (mois) est compris entre 01 et 12.
  - DD (dpt.) est compris entre 01 et 95 ou 99 (étrangers).
  - Présence du tiret.
  - CC (clé) doit correspondre au numéro.
- Indique si le numéro est valide ou quelle partie est incorrecte.

TD (3/3)

nir15

- Indique si le numéro est valide ou quelle partie est incorrecte.
- Exemples :

```
NIR15 ? 2921059286404-08
Clé incorrecte

NIR15 ? 292105928640410
Format NIR15 incorrect !

NIR15 ? 3921059286404-10
Sexe incorrect !

NIR15 ? 2921059286404-10
NIR15 correct
```

Séquences : tuples (classe **`tuple`**)

- Séries immutables et ordonnées de données hétérogènes.
- Immutables... mais peuvent contenir des données mutables.
- Exemple :

```
t = 1, 'toto', 2.5          #Packing
print(t)                    #(1, 'toto', 2.5)
a, b, c = t                 #Unpacking
print(a, b, c)              #1 toto 2.5
print(t[1])                 #toto
print(t[1:2])               #('toto',) virgule oblig.
t += 3, (4, 5)              #Append et imbrication
print(t)                    #(1, 'toto', 2.5, 3, (4, 5))
```

Séquences : listes (classe `list`)

- Séries mutables et ordonnées de données hétérogènes.
  Traditionnellement utilisées pour des données homogènes.
- Exemple :

```
liste = [1, 2, 3]
a, b, c = liste                 #Unpacking
print(a, b, c)                  #1 2 3
print(liste[1])                 #2
print(liste[1:2])               #[2] Slicing
liste += [4, 5, 6]              #Append
liste[1:3] = [7, 8]             #OK, mutable
print(liste)                    #[1, 7, 8, 4, 5, 6]
liste = [4, [1, 2, 3], 5] #Imbrication
```

Séquences : ranges (classe **`range`**)

- Séries immutables de nombres.
- Essentiellement utilisées en conjonction avec le constructeur **`list()`**.
- Egalement utilisées pour les boucles indexées (voir plus loin).
- Exemples :

```
list(range(5))          #[0, 1, 2, 3, 4]
list(range(2, 5))       #[2, 3, 4]
list(range(2, 5, 2))    #[2, 4] (pas (step) de 2)
```

- Sans **`list()`**, **`range()`** retourne un iterateur. Voir plus loin.
- La fonction *builtin* **`range()`** est un alias du constructeur de la classe.

Opérations communes aux séquences

`str, tuple, list, range...`

`in`	Présence d'une valeur scalaire (ou d'une séquence pour `str`).
`not in`	Absence d'une valeur scalaire (ou d'une séquence pour `str`).
`+`	Concaténation (sauf `ranges`).
`*`	Répétition (sauf `ranges`).
`[i]`	Indexation.
`[a:b]`	*Slicing*.
`[a:b:step]`	*Slicing* avec pas.
`len(seq)`	Nombre d'éléments.
`min(seq)` `max(seq)`	Minimum ou maximum.

Dictionnaires (class `dict`)

- Séries mutables non-ordonnées de données hétérogènes indexées par des clés (table de hachage).
- Exemples :

```
notes = {'toto':12, 'lulu':15}
print(notes)                  #{'toto':12, 'lulu':15}
print(notes['toto'])          #12
dico={}                       #Dictionnaire vide
print(list(notes.keys()))     #['lulu', 'toto']
```

Méthodes communes aux mutables

`list, dict...`

`.append(x)`	Ajoute un élément à la fin de la séquence.
`.clear()`	Vide la séquence.
`.copy()`	Copie (équivalent à `[:]`).
`.extend(seq)`	Ajout d'une séquence à la fin d'une séquence.
`.insert(i, x)`	Insert un élément à un indice donné.
`.remove(x)`	Supprime de la séquence le premier élément égale à `x`.
`.reverse()`	Inverse l'ordre de la séquence (sans effet dans `dict`).

Jeux (class `set`) (1/2)

- Séries immutables non-ordonnées de données dédoublonnées.
- Indexation, concaténation et imbrication non supportées.
- Exemples :

```
jeu = {'toto', 'lulu', 'toto'} #Dédoublonnage auto.
print(jeu)                     #{'lulu', 'toto'}
a, b = jeu                     #Unpacking
print(a, b)                    #lulu toto
print(len(jeu))                #2
```

Jeux (class `set`) (2/2)

```
j1 = set('abracadabra') #{'a', 'r', 'b', 'c', 'd'}
j2 = set('tralala')     #{'a', 'r', 't', 'l'}
print(j1 - j2)          #{'c', 'b', 'd'} Différence
print(j1 & j2)          #{'a', 'r'} Intersection
print(j1 | j2)          #{'l', 'a', 'c', 'b', 'd',
                          'r', 't'} Union
print(j1 ^ j2)          #{'l', 'c', 'b', 'd', 't'}
                         Différence symétrique
```

Parcourir `str, tuple, list, set`

- Exemple :

```
liste = [4, 5, 6]
for i in liste:
  print(i)
#4
#5
#6
```

Parcourir `str, tuple, list, set`

avec un index

- Exemple (`enumerate()` est une fonction *builtin*) :

```
liste = ['toto', 'lulu', 'lili']
for i, prenom in enumerate(liste):
  print(i, prenom)
#0 toto
#1 lulu
#2 lili
```

Parcourir `dict`

- Exemple :

```
notes = {'toto':12, 'lulu':15}
for prenom, note in notes.items():
  print(f"{prenom} : {str(note)}/20")
#lulu : 15/20
#toto : 12/20
```

Boucles indexées

- Exemple :

```
liste = [4, 5, 6]
for i in range(0, len(liste)):
  print(liste[i])
#4
#5
#6
```

Boucles `while`

- Exemple :

```
n = 1
while n < 4:
  print(n)
  n += 1
#1
#2
#3
```

Boucles conditionnelles

- **`break`**
  Provoque la sortie de la boucle en cours.
- **`continue`**
  Provoque le saut à l'itération suivante.
- **`else`**
  A la suite d'une boucle, sera toujours exécuté SAUF en cas de **`break`**.

Compréhensions de listes

- Crée une liste en faisant subir un traitement aux éléments d'une autre.
- S'applique de façon similaire aux dictionnaires et aux jeux.
- Exemple :

```
liste = [4, 5, 6]
liste2 = [i ** 2 for i in liste]
print(liste2) #[16, 25, 36]
```

Comparaison de séquences

`str, tuple, list, range...`

- Uniquement possible entre séquences de même type.
- Exemples d'assertions **vraies** :

```
(1, 2, 3) < (1, 2, 4)

[1, 2, 3] < [1, 2, 4]

'ABC' < 'C' < 'Lulu' < 'Toto'

(1, 2, 3, 4) < (1, 2, 4)

(1, 2) < (1, 2, -1)

(1, 2, 3) == (1.0, 2.0, 3.0)
```

TD

fibo300

- Ecrivez un programme qui affiche les termes inférieurs à 300 de la suite de Fibonacci.
- RAPPEL
  La suite de Fibonacci est définie ainsi :

$$\mathcal{F}_1 = 1$$

$$\mathcal{F}_2 = 1$$

$$\mathcal{F}_n = \mathcal{F}_{n-1} + \mathcal{F}_{n-2}$$

TD

fibo20

- Ecrivez un programme qui affiche les 20 premiers termes de la suite de Fibonacci (indices 1 à 20).

TD

verlan

- Ecrivez un programme qui demande la saisie d'une chaîne puis l'affiche inversée. Exemple : CAMION devient NOIMAC.

Fonctions

- Peuvent retourner ou non un résultat.
- Peuvent être récursives.
- Exemple :

```
def add(a, b):
  return a + b

c = add(3, 5)
print(c) #8
```

Portée des variables (1/2)

- Une variable définie en dehors d'une fonction a une portée globale.
- Une variable définie dans une fonction a une portée locale.
- Les variables locales masquent les variables globales de mêmes noms. Le mot-clé **`global`** permet de contrer cela.
- Exemple :

```
def f():
  global g1
  g1 = 'f-global-g1'
  g2 = 'f-g2'
  g3 = 'f-g3'
  print(g1, g2, g3)
```

Portée des variables (2/2)

```
g1 = 'script-g1'
g2 = 'script-g2'
f()             #f-global-g1 f-g2 f-g3
print(g1, g2) #f-global-g1 script-g2
print(g3)       #Erreur
```

TD

recMax

- Les fonctions Python acceptent la récursivité...
  Mais à combien de niveaux maximum ?

Fonctions

Arguments par défaut

- Exemple :

```
def supprimer(id, forcer = False):
   ...

supprimer(3)
supprimer(8, True)
```

Fonctions

Arguments nommés

- Exemple :

```
def add(colonne, rangee, dim = 1):
  ...

add(rangee = 3, colonne = 5)
add(rangee = 2, colonne = 4, dim = 2)
```

Fonctions

Rest parameters (1/2)

- **`*args`**

  Il reçoit tous les arguments positionnés dans un tuple.
  Exemple :

```
def supprimer(*ids, forcer = False):
  ...

supprimer(3, 8, 5, 4)
supprimer(3, 8, forcer = True)
```

Fonctions

Rest parameters (2/2)

- **`**args`**

  Il reçoit tous les arguments nommés dans un dictionnaire.
  Exemple :

```
def add(dim = 1, **args):
  ...

add(rangee = 2, colonne = 4, dim = 2)
```

Fonctions

Passage par référence ou par valeur ?

- Le passage a toujours lieu par référence mais les variables immutables ne sont pas modifiées donc tout se passe pour elles comme un passage par valeur.

TD

passage

- Ecrivez un programme qui démontre les différences entre le passage d'un argument mutable et d'un autre immutable.

Modules

Présentation

- Tout fichier `.py` constitue un module (même le programme principal).
- Un module peut contenir :
  - Des déclarations de variables.
  - Des fonctions à la racine.
  - Des classes (voir plus loin).
  - Un script à la racine.
    Il sera exécuter lors de l'importation.
  - Un programme principal.
    A précéder d'un test conditionnel pour détecter l'appel façon programme principal.

Modules

Importation d'un module (lui-même)

- Seul le nom du module est ajouté à l'espace de noms.
  Exemple :

```
import math
...

print(math.sqrt(16)) #4.0
print(math.pi)       #3.141592653589793
```

Modules

Importation sélective d'éléments d'un module

- Chaque élément importé est ajouté à l'espace de noms.
- Exemple :

```
import math
from math import sqrt
...

print(math.sqrt(25)) #5.0 Lourdeur inutile !
print(sqrt(16))      #4.0 Plus pratique !
print(math.pi)       #3.141592653589793
print(pi)            #Erreur !
```

Modules

Importation globale d'un module

- Forte pollution de l'espace de noms : danger de collisions !
- Exemple :

```
from math import *
```

TD

carMag

- Ecrivez un programme qui demande une dimension (supérieure ou égale à 2) puis affiche une solution de carré magique dans la dimension saisie ou un message s'il n'y a pas de solution.
- RAPPEL
  Un carré magique de dimension N est un quadrillage dans lequel chaque rangée, chaque colonne et les 2 grandes diagonales contiennent les nombres de 1 à N sans doublon.

POO

Rappels (1/2)

- Un objet contient des membres.
- Deux types de membres : propriétés et méthodes.
  Une propriété est une information.
  Une méthode est une capacité d'action.
- L'ensemble des propriétés constitue l'état de l'objet.
- Une classe peut créer des objets dits instances de la classe.
  Dans chaque classe, un constructeur permet de créer des instances.
- Chaque instance d'une même classe possède ses propres valeurs de propriétés et partage les méthodes avec les autres instances.

POO

Rappels (2/2)

- Il existe des propriétés de classe dont les valeurs sont communes à l'ensemble des instances. Elles sont dites statiques.
- Il existe des méthodes de classe qui n'agissent pas sur les instances mais sur la classe elle-même. Elles sont dites statiques.

POO

Classes

- Exemple :

```
class Vehicule:
  pass #Le mot-clé pass ne fait rien.

v = Vehicule() #Crée une instance de Vehicule.
```

POO

Propriétés dynamiques

- Des propriétés peuvent être ajoutées dynamiquement. Exemple :

```
v1 = Vehicule()
v1.couleur = "rouge"
v2 = Vehicule()
v2.nbRoues = 4
print(v1.couleur) #rouge
print(v1.nbRoues) #Erreur !
print(v2.couleur) #Erreur !
print(v2.nbRoues) #4
```

- **Maintenant, la classe Vehicule et ses instances v1 et v2 ne possèdent pas les mêmes propriétés. Technique dangeureuse !**

POO

Variables de classe

- Déclarées à la racine de la classe.
- Accessibles via le nom de la classe.
- Exemple :

```
class Vehicule:
  nb = 0

v = Vehicule()
Vehicule.nb += 1
print(Vehicule.nb) #1
```

POO

Constructeur

- Il se nomme obligatoirement **`__init__()`** et reçoit **implicitement en premier paramètre** l'instance en cours de création (mot-clé **`this`** dans la plupart des langages) libre de nom en Python et traditionnellement appelée **`self`**. Exemple :

```
class Vehicule:
  nb = 0
  def __init__(self):
    Vehicule.nb += 1

v1 = Vehicule()
v2 = Vehicule()
print(Vehicule.nb) #2
```

POO

Variables d'instance

- Obligatoirement déclarées dans le constructeur.
- Exemple :

```
class Vehicule:
  nb = 0
  def __init__(self, couleur = 'noir'):
    self.couleur = couleur
    Vehicule.nb += 1

v1 = Vehicule("rouge")
v2 = Vehicule()
print(v1.couleur) #rouge
print(v2.couleur) #noir
```

POO

Méthodes de classe (1/2)

- Précédées du modificateur (*decorator*) `@classmethod`.
- Reçoivent implicitement en premier argument la classe dont le nom est libre mais traditionnellement `cls`.
- Exemple diapo suivante...

POO

Méthodes de classe (2/2)

```
class Vehicule:
  nb = 0
  def __init__(self):
    Vehicule.nb += 1
  @classmethod
  def denombrer(cls):
    print(f"Nb. de véhicules : {Vehicule.nb}")

v1 = Vehicule()
v2 = Vehicule()
Vehicule.denombrer() #Nb. de véhicules : 2
```

POO

Méthodes d'instance

- Reçoivent implicitement en premier argument l'instance en cours de création traditionnellement appelée **`self`**.
- Exemple :

```
class Vehicule:
  def __init__(self, couleur = 'noir'):
    self.couleur = couleur
  def afficher(self):
    print(f"Couleur : {self.couleur}")

Vehicule("rouge").afficher() #Couleur : rouge
Vehicule("blanc").afficher() #Couleur : blanc
```

TD

boulangerie1

- A l'aide du programme principal, de la sortie et des indications suivantes, écrivez le code de l'ensemble des classes.

  Main : *boulangerie1/boul1.py*
  Sortie : *boulangerie1/sortie1.txt*

- Les boulangers
  - Ils ont un prénom.
  - Ils fabriquent des produits.
- Les produits
  - Ils ont un nom, un prix de revient et un prix de vente.

TD

boulangerie2

- Modifiez le code de l'ensemble des classes.

  Main : *boulangerie2/boul2.py*
  Sortie : *boulangerie2/sortie2.txt*

- Les boulangers
  - Ils entretiennent une liste de leurs fabrications.
  - Ils éditent un bilan des fabrications.

TD

boulangerie3

- Modifiez le code de l'ensemble des classes.

  Main : *boulangerie3/boul3.py*
  Sortie : *boulangerie3/sortie3.txt*

- Les vendeuses
  - Elles vendent des produits.
  - Elles entretiennent une liste de leurs ventes.
- Les boulangers
  - Ils embauchent des vendeuses.
  - Ils entretiennent une liste de leurs vendeuses.
  - Ils éditent un bilan des fabrications et des ventes.

POO

Rappel : héritage

- Une classe enfant peut hériter d'une classe parente.
- Une classe enfant hérite les membres de son parent.
- Une classe enfant peut comporter de nouveaux membres.
- Une classe enfant peut redéfinir (*override*) les méthodes de sa classe parente, c'est à dire remplacer une méthode parente par une autre de même signature (même nom, mêmes paramètres).
- L'héritage correspond à une relation EST UN entre enfant et parent ou encore, à une relation de spécialisation/généralisation.
- Certains langages (dont Python) permettent l'héritage multiple (plusieurs parents). D'autres préfèrent les interfaces (Java...).

POO

Duck-typing

- Dans les langages sans typage statique, le polymorphisme n'existe pas puisqu'il n'y a pas de type déclaré.
- Le duck-typing garantit simplement que si une méthode donnée existe dans une classe ou dans l'une de ses classes parentes, alors toute instance de cette classe pourra appeler cette méthode avec succès. C'est la méthode de la classe donnée ou, à défaut, de son plus proche parent qui sera exécutée.

POO

Héritage simple (1/2)

- La fonction *builtin* **`super()`** permet d'accèder explicitement aux membres des parents.
- Exemple diapo suivante...

POO

Héritage simple (2/2)

```
class Media:
  def __init__(self, titre = ''):
    self.titre = titre

class Livre(Media):
  def __init__(self, titre = '', nbPages = ''):
    super().__init__(titre)
    self.nbPages = nbPages

livre = Livre("Millenium", 350)
print(f"{livre.titre} : {livre.nbPages} pages")
#Millenium : 350 pages
```

POO

Appel implicite au constructeur parent

- En l'absence (et seulement en l'absence) de méthode **`__init__()`**, il y a appel implicite au constructeur parent **avec transmission des arguments.** Exemple :

```
class Parent:
  def __init__(self, prenom = ''):
    print(f"Je suis la mère {prenom}")

class Enfant(Parent):
  pass

Parent("Michèle") #Je suis la mère Michèle
Enfant("Noël")    #Je suis la mère Noël
```

POO

Redéfinition (*overriding*)

- Toute méthode peut être redéfinie dans une sous-classe.
  Exemple :

```
class Animal:
  def crier(self):
    print("???!")

class Chien(Animal):
  def crier(self):
    print("Ouah!")

Chien().crier() #Ouah!
```

POO

Héritage multiple (1/2)

- L'ordre compte : attention aux ambiguïtés !
  Exemple :

```
class Animal:
  pass

class Volant:
  pass

class Oiseau(Animal, Volant):
  pass

class Coccinelle(Volant, Animal):
  pass
```

POO

Héritage multiple (2/2)

```
class Mouette(Oiseau, Volant):          #=(Animal,Volant,Volant)
  pass                                  #=(Animal,Volant)

class Pigeon(Volant, Oiseau):           #=(Volant,Animal,Volant)
  pass                                  #Erreur, ambiguïté !
```

POO

Interroger le type d'un objet (1/2)

- Fonctions *builtins* :

`type(obj)`	Retourne une représentation `str` de la classe de `obj`.
`isinstance(obj, cls)`	Retourne `True` si `obj` est du type `cls` ou d'un de ses parents.

POO

Interroger le type d'un objet (2/2)

```
class A:
  pass

class B(A):
  pass

print(type(A()))           #<class '__main__.A'>
print(type(B()))           #<class '__main__.B'>
print(isinstance(A(), A)) #True
print(isinstance(A(), B)) #False
print(isinstance(B(), A)) #True
print(isinstance(B(), B)) #True
```

POO

Egalité entre objets

- Exemple :

```
class A:
  pass

a1 = A()
a2 = a1
a3 = A()
print(a1 is a2)         #True
print(a1 is a3)         #False
print(a2 is a3)         #False
print(type(A()) is A)   #True
```

TD

boulangerie4 (1/2)

- Modifiez le code de l'ensemble des classes.

  Main : *boulangerie4/boul4.py*
  Sortie : *boulangerie4/sortie4.txt*

- Les boulangeries
  - Elles ont un nom.
  - Elles embauchent des boulangers (même pâtissiers) et des vendeuses.
  - Elles entretiennent une liste de leurs employés.
  - Elles éditent un bilan des fabrications et des ventes.

TD

boulangerie4 (2/2)

- Les pâtissiers
    - Ils sont des boulangers spécialisés qui peuvent fabriquer des produits de boulangerie ou des patisseries.
- Les pâtisseries
    - Elles sont des produits spécialisés.
    - Elles peuvent être ou non au beurre.

POO

Méthodes magiques

- Elles permettent d'implémenter des fonctionnalités particulières et de redéfinir des opérateurs ou des méthodes.
  Exemples :

`__lt__(self, other)`	Redéfinit l'opérateur **<** utilisé par la fonction *builtin* `sorted()`.
`__eq__(self, other)`	Redéfinit l'opérateur **==**.
`__add__(self, other)`	Redéfinit l'opérateur **+**.
`__and__(self, other)`	Redéfinit l'opérateur `and`.
`__or__(self, other)`	Redéfinit l'opérateur `or`.
`__str__(self)`	Redéfinit la méthode `str()` utilisée par `print()`.

TD

boulangerie5 (1/2)

- Dans la classe **`Vendeuse`** de *boulangerie4* :
  - Implémentez **`__str()__()`** pour afficher le prénom.
  - Implémentez la méthode **`__lt__()`** pour classer les vendeuses selon leur CA.
- Vérifiez le fonctionnement avec le programme de la diapo suivante...

TD

boulangerie5 (2/2)

```
croissant = Produit("croissant", 0.15, 1.10)
painDeMie = Produit("pain de mie", 0.35, 2.50)
v1 = Vendeuse("v1")
v2 = Vendeuse("v2")
v3 = Vendeuse("v3")
v1.vendre(croissant, 50)
v1.vendre(painDeMie, 10)  #(CA = 80)
v2.vendre(painDeMie, 30)  #(CA = 75)
v3.vendre(croissant, 100) #(CA = 110)
vendeuses = sorted([v1, v2, v3])
for vendeuse in vendeuses:
  print(vendeuse, end = ", ") #v2, v1, v3
```

Exceptions

Capture

- Syntaxe :

```
try:
   ... code à tenter ...
except [classeException as erreur,...]:
   ... traitement de l'exception ...
else:
   ... code si aucune exception ...
```

- Un seul try.
  Zéro, un ou plusieurs except.
  Zéro ou un else.

- Sans classe précisée, toutes les erreurs sont capturées !

Exceptions

Exemples d'exceptions *builtins*

`ZeroDivisionError`	Division par zéro.
`NameError`	Identifiant indéfini.
`IndexError`	Index indéfini dans une séquence.
`KeyError`	Clé indéfinie dans un dictionnaire.
`TypeError`	Types incompatibles.
`AttributeError`	Membre indéfini.

Exceptions

Capturer de la plus précise à la plus générale

- MAUVAIS exemple :

```
try:
  x = n / d
except Exception as err:  #Capturera TOUT
  print(err)
except ZeroDivisionError: #Ne capturera RIEN
  print("Division par zéro !")
else:
  print("Résultat :", x)
```

Exceptions

Déclenchement

- Exemple :

```
raise NameError #Déclenche l'exception.
```

Exceptions

Classe d'exception personnalisée

- Possible de créer une classe d'exception personnalisée en étendant la classe Exception.

TD

boulangerie6

- Dans la classe Vendeuse de *boulangerie5*, créer une classe d'exception personnalisée.
- Quand une vendeuse vend une quantité inférieure ou égale à zéro, l'exception doit se déclencher et afficher :

```
BoulangerieException: Quantité incorrecte
```

- Vérifiez le fonctionnement avec le programme suivant :

```
p = Produit('p', 1.0, 2.0)
v = Vendeuse('v')
v.vendre(p, 0)
```