TechnofuturTic - Stage Python & Raspberry Pi

Python

Florence Blondiaux
Jean-Martin Vlaeminck

Université Catholique de Louvain

06/08/18 au 10/08/18

Section 1

Présentation

Langage de programmation

Définition Wikipédia LANGAGE

Le langage est la capacité d'exprimer une pensée et de communiquer au moyen d'un système de signes (vocaux, gestuel, graphiques, tactiles, olfactifs, etc.)...

Simplification: Moyen de communication entre deux entités (personnes, machines, ...)
Exemple: français, néerlandais, anglais, espagnol,...

Pourquoi Python?

- Langage de haut niveau → pas de programmation avec des 0 et 1 comme dans les films
- Multiplateforme : Windows, OS X, Linux,...
- Gratuit
- Programmes clairs et structurés
- Pratique pour les débutants

Installation et fonctionnement

Démonstration en direct.
Installer un environement de développement.

Section 2

Hello World!

Premier programme

- Le célèbre "Hello World!" ("Bonjour Monde!", en Français)
- Un programme simple qui affiche un petit texte à l'écran
- Permet d'apprendre à exécuter un programme Python

Le programme

Un programme très très (très ?) compliqué :

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

Explications

La fonction `print()` permet d'afficher à la console tout ce qui se trouve dans la parenthèse que la suit.

A vous de jouer !

Mission

Modifier le code pour que l'ordinateur affiche "Bonjour maman"

Commentaires

Outils essentiels utilisés pour donner des indications à la personne qui lit votre programme. Ces phrases sont ignorées lors de l'exécution du programme !

En python : On commente une phrase en la précédant du symbole # et on commente un texte en le précédant de """ et en le terminant avec """[1].

[1] Attention pas d'accents ni de symboles chelous dans les commentaires !

Le programme

Un programme commenté :

```
"""Ce superbe programme vous est offert par
les entreprises Moumal. Attention il doit
etre utilise avec moderation ! L abus de
python est dangereux pour la sante, ..."""
print("salut") #Imprime salut
```

A vous de jouer !

Mission

Commenter le code de l'exercice précédent en précisant **l'auteur du code**, **la date de création**, **le lieu de création** et expliquer ce que fait l'instruction `print` pour quelqu'un n'ayant jamais fait de Python.

Section 3

Variables

Variables

Qu'est ce qu'une variable ? Les variables sont des symboles qui associent un nom à une valeur. La valeur est stockée dans la mémoire de l'ordinateur et le nom permet d'y accéder et/ou de modifier la valeur en tout temps. Une variable possède toujours un **type**. Ce dernier renseigne sur la nature de la valeur stockée dans la variable.

En python

Les noms de variables ne peuvent pas commencer par un chiffre, ni contenir d'espaces, ni contenir d'accents.

```
nom_de_variable = valeur
#Exemples de variables correctes
variable1 = 1 #nom : variable1, valeur : 1
b = "salut"   #nom : b, valeur : "salut"
variable68452 = 0.50
variable_au_nom_tres_long = True
variable_lettre = 'a'
```

Types

Type	Description
`boolean`	Représente *True* (vrai) ou *False* (faux)
`str`	Représente une chaîne de caractères
`int`	Représente un entier
`float`	Représente un nombre à virgule

Listing 1: Types de variables

```
niveau = "E"  #variable de type str
chaises = 12  #variable de type int
sonActif = False  #variable de type boolean
prixJeu = 12.50  #variable de type float
totalVoitures = 3453454  #variable de type int
```

Changer la valeur

Listing 2: Réassignation de variables

```
a = 2
print(a) #Affiche 2
a = 1
a = 4
print(a) #Affiche 4
a = "arbre" #Que se passe-t-il ?
print(a) #Affiche "arbre"
```

Opérations

Opération	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division
**	Puissance
%	Modulo

```
a = 4 + 6 #a vaut 10
a = ((a + a)/2)+1 #a vaut 11
b = a / 2 #b vaut 5
c = a**2 + b #c vaut 11^2 + 5 = 121 + 5 = 126
```

A vous de jouer !

Mission 1

Créer un code qui calcule et imprime successivement

1. $a = 124^3 + 27^2 + 7^3$
2. $b = 2589\%60$
3. $c = \frac{a^2 - b^3}{a}$
4. $a + b + c$

Mission2

Quelles sont les valeurs des différentes variables après l'exécution de cette partie de code?

```
x = 10
x = x * 2
y = x - 10
z = y / x
```

Section 4

Strings

Strings

Un String est le type qui représente un texte. Tout texte entouré de guillemets est considéré comme un `String`.

```
#Exemples de Strings
v1 = "Ceci est une phrase"
v2 = ""
v3 = "3" #Est un String et non un int !
v4 = "42.5"
v5 = "qpdjnqpuifnsfvsnvsdv5s1vd1s5v15xv4s5d"
print(type(v1))#Imprime str (String)
print (v3/3) #Que se passe t il ?
```

Accès aux lettres d'un String

Chaque lettre du String est numérotée par ordre croissant (commençant par zéro). Ainsi le String "Hello" possède la lettre *H* en 0, la lettre *e* en 1, etc... On accède à une lettre d'un `String` de la manière suivante :

```
# Accede a la lettre d'un String
nom_du_string[numero_de_la_lettre]
#Exemples
my_string = "Bonjour"
lettre_1 = my_string[0] #lettre_1 vaut 'B'
lettre_4 = my_string[3] #lettre_4 vaut 'j'
```

Opérations sur les Strings

Voilà une liste d'opérations très utiles sur les Strings :

Opération	*Description*
len(nom_string)	Calcule la longueur d'un String
nom_string.lower()	Retourne le String en minuscule
nom_string.upper()	Retourne le String en majuscule
str(nom_variable)	Cast en String
nom_string1+nom_string2	Retourne les 2 Strings concaténés

A vous de jouer !

Mission 1

Concaténer les `Strings` "Hakuna Matata", "Mais quelle ", "phrase magnifique !" et affichez le résultat en majuscules ainsi que sa longueur.

Mission 2

Caster les Strings a = "2" et b = "125.2" en nombre pour calculer a/b.[a]

[a] a et b doivent être impérativement définis en String au départ !

Interaction avec l'utilisateur

On peut demander à l'utilisateur de rentrer des informations avec la fonction `input()`. Par défaut, cette fonction convertit toutes les données entrées par l'utilisateur en `String` mais on peut imposer de recevoir un type en particulier en *castant* la réponse de l'utilisateur dans le type désiré.

Exemple

```
answer = input("ce_qu_on_affiche_a_l_utilisateur")
#Exemple
nom = input("Quel est votre nom ?")
prenom = input("Quel est votre prenom ?")
age = int(input("Quel est votre age ?"))
print("Bonjour " + nom + " " + prenom + " vous etes age de "
+ str(age) + " ans!")
```

A vous de jouez !

Mission

Faites un convertisseur qui demande à l'utilisateur un nombre en kilomètres et qui lui renvoie ce chiffre converti en miles en le remerciant d'avoir utilisé votre programme. *Aide : 1 km = 0.621 mile*

Section 5

Conditions

Outils de comparaison

On peut comparer des variables entre elles. Le résultat de cette comparaison **est un booléen**.

Opération	Description
==	Egalité
! =	Différent
>	Plus grand
>=	Plus grand ou égal
<	Plus petit
<=	Plus petit ou égal

Exemples

```
a = 5 > 4 # a vaut True
b = 0.5 >= 1 # b vaut False
c = 1 > "salut" # Que se passe t il ?
```

Opérateurs logiques

On construit des expression booléennes complexes en en combinant plusieurs grâce aux opérateurs logique.

a OU b	a or b
a ET b	a and b
NON a	not a

Priorité : `()`, `not`, `and`, `or`

Exemples

```
a = True
b = False
c = a or b #c est vrai
d = a and b #d est faux
e = not b and a #e est vrai
f = (not (a and b)) or (not ((not d) and (a or e)))
#f vaut True
```

A vous de jouer !

Mission

Evaluez les expressions booléennes suivantes.

```
bool_one = False or not True and True
bool_two = False and not True or True
bool_three = True and not (False or False)
bool_four = not not True or False or not True
bool_five = False or not (True and True)
```

Le if

Permet d'exprimer une condition sous laquelle on va exécuter une partie du code. La condition doit être une expression booléenne. ATTENTION à ne pas oublier l'indentation.

```
a = 10
if a > 5:
    print("Cette instruction est executee")
print("fin du code")
```

```
a = 3
if a > 5:
    print("Cette instruction n'est pas executee")
print("fin du code")
# "fin du code" est toujours affiche
# puisqu'il n'est pas dans la condition
```

Le else

Permet d'exprimer une alternative si la condition n'est pas vraie.

```
a = 10
if a < 5:
    print("Cette instruction n'est pas executee")
else:
    print("Cette instruction est executee")
print("fin du code")
```

Le elif

Enfin, on peut tester plusieurs conditions à la fois. Dès qu'une condition est vérifiée, les autres ne sont plus évaluées.

```
x = 200
if x > 100:
    print("A")
elif x > 10:
    print("B")
else:
    print("C")
# Seul 'A' est imprime, pas 'B' (et forcement pas 'C')!!!
```

Le code mort

Lorsque vous utilisez des conditions, faites attention à ne pas créer du *code mort*, c'est-à-dire du code qui ne sera jamais exécuté.

```
if False:
    print("Ce code n'est jamais execute!")
```

```
if a > 5 and a <= 5 :
    print("Ce code n'est jamais execute!")
```

Conditions plus complexes...

On peut très bien combiner des conditions en les imbriquant les unes dans les autres.

```
x = int(input("Entrez un nombre"))
if x >= 0:
    print("Nombre positif")
    if x > 1000000:
        print("C'est un tres grand nombre!")
else:
    print("Nombre negatif")
```

Mission

- Définissez une variable `annee`, et imprimez "vrai" si l'année est bissextile. (**Aide :** une année est bissextile si elle est divisible par 4 et non divisible par 100, OU si elle est divisible par 400.)
- Ecrivez un code qui demande à l'utilisateur d'entrer un chiffre et imprime "Félicitations, votre chiffre est un multiple de 7" si le chiffre est divisible par 7. Et "Désolé, votre chiffre n'est pas un multiple de 7" sinon.

Section 6

Récapitulatif

A vous de jouer!

Mission

Affichez à l'écran "Timon ou Pumbaa?"
Le but du programme est de vérifier que l'utilisateur écrit "Timon", et il n'a que deux essais.
Si le deuxième essai est faux, on affiche "Tu prends la porte".
Si l'utilisateur écrit Timon en un ou deux essais, on affiche "Bon choix!"
Attention, vous ne pouvez utiliser que des conditions!
Commentaires obligatoires!

Section 7

Boucles

La boucle

"Tant que cette condition est vraie, j'exécute le code suivant"

Boucle while

Ecriture `while expr_booleenne:`

1. On évalue l'expression booléenne
2. Si l'expression est **False** : on ignore le code indenté qui suit le **while**.
3. Si l'expression est **True** : on exécute le code indenté qui suit **while** et retour à l'étape 1.

```
"""Affiche tous les nombres de la table de 9 plus petits
que 100"""
i = 0
while i <= 100:
        print("i vaut " + str(i))
        i = i+9 # Mise a jour de i
print("Fini")
```

Mise à jour

Attention à bien mettre à jour les variables dans l'expression booléenne ! Dans le cas contraire, on a une boucle qui ne s'arrête jamais : **une boucle infinie**. Elles peuvent littéralement tuer votre programme !

```
i = 0
while i < 10:
    print("Je suis dans la boucle")
print("Ce message ne s'affichera jamais")
```

Boucle for

Permet d'appliquer un code (*indenté*) à chaque élément d'une structure de données[2]. Elle commence toujours par le premier élément de la structure et finit par le dernier.

Ecriture `for i in ma_structure:`

Ici `i` est appelé *itérateur*, c'est-à-dire que c'est lui qui prendra successivement la première valeur de la structure de données, puis la seconde, et ainsi de suite.

[2] Par exemple un String ou une liste

Exemples

```
#Utilise une boucle for pour imprimer un string
s = "Hello World"
for i in s:
    print(i)
print("done")
```

```
#Utilise une boucle for pour iterer les elements de 0 a 100
for i in range(0,101):
    print(i)
print("done")
```

Le break

Instruction pour sortir à tout moment d'une boucle. On la met à l'endroit où on souhaite en sortir.

```
i = 0
while(i<100):
    if i != 50:
        print(i)
        i = i+1
    else:
        break #On sort quand i == 50
```

Attention : un `break` est considéré comme une sortie "sale" d'une boucle. On ne l'utilise donc que lorsque cela est absolument nécessaire !

A vous de jouer !

Mission 1

Améliorez votre convertisseur pour qu'il convertisse les unités de l'utilisateur jusqu'à ce que celui entre le mot "stop".

Mission 2

Implémentez un programme qui prend un texte en entrée (fourni par l'utilisateur) et imprime chaque voyelle du texte. (Essayez avec un texte très long trouvé sur Internet...)

Mission 3

Implémentez un programme qui prend un texte en entrée (fourni par l'utilisateur) et qui imprime **True** si le texte contient un palindrome et **False** sinon. (**Aide :** un palindrome est un mot qui peut se lire dans les 2 sens. *Exemple* : kayak, abcba, ...)

Section 8

Listes

Créer une liste

Une liste est un ensemble d'objets. On peut y stocker des variables de n'importe quel type.

```
jeuxVideos = ['HeartStone', 'LoL', 'WoW', 'ClubPenguin']
chiffres = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
bazar = ['Coucou', 42, 'petite', 0.02, 'perruche', True]
```

Accès dans une liste

L'accès aux éléments d'une liste est similaire à l'accès aux lettres d'un String (= cas particulier de liste!).

```
bazar = ['Coucou', 42, 'petite', 0.02, 'perruche', True]
print(bazar[0])    # Imprime "Coucou"
print(bazar[5])    # Imprime "True"
print(bazar[2:4]) # Imprime "['petite', 0.02]"
```

Modification de listes

En python, une liste n'est pas immuable : on peut changer/supprimer/ajouter un ou plusieurs éléments.

```
my_list = ['a', 'b', 'c', 'd', 'e']
#Modification
my_list[3] = 'f'      #['a', 'b', 'c', 'f', 'e']
#Suppression
del my_list[3]        #['a', 'b', 'c', 'e']
my_list.remove('e') #['a', 'b', 'c']
#Ajout
my_list.append('d') #['a', 'b', 'c', 'd']
my_list + ['e']       #['a', 'b', 'c', 'd', 'e']
```

Opérations sur les listes

Voici quelques opérations communes et pratiques sur les listes.

```
my_list = ['a', 'b', 'c', 'd', 'e']
#Longueur de la liste
longueur = len(my_list)  #longueur vaut 5
#Presence d un element
'd' in my_list #vaut True
'f' in my_list #vaut False
#Parcourir une liste
for x in my_list : print (x) #imprime abcde
#Retourner une liste
my_list.reverse() #my_list vaut ['e', 'd', 'c', 'b', 'a']
#Mettre une liste dans l ordre croissant
my_list.sort() #my_list vaut ['a', 'b', 'c', 'd', 'e']
```

A vous de jouer !

Mission 1

Créez une liste de Strings contenant au moins 5 fruits et légumes, puis triez la dans l'ordre alphabétique.

1

Mission 2

Ajoutez à votre liste les éléments suivants **s'ils ne sont pas déjà présents** : ['banane', 'tomate', 'rutabaga', 'reine-claude', 'patate']

2

Mission 3

Éliminez de votre liste tous les éléments dont la première lettre commence par la lettre **p**.

3

Section 9

IO

Ouverture d'un fichier

Pour manipuler un fichier, il faut toujours commencer par l'ouvrir !
Ouverture avec la fonction `open(nom_du_fichier, Permissions)`. Où nom_du_fichier est simplement le nom du fichier[3] que l'ont veut ouvrir et Permissions décide de ce qu'on peut en faire.

Permissions :

- "r" pour lecture
- "w" pour écriture
- "r+" pour lecture et écriture

[3] Le fichier doit exister et se trouver dans le répertoire de anaconda !

Ecriture dans un fichier

On **ouvre** d'abord le fichier dans lequel on souhaite écrire avec la permission d'écriture. Et on stocke la valeur de retour de la fonction `open` dans une variable [4].
Ecriture avec la fonction `write()` :
variable_de_fichier.`write("String_de_ce_qu_on_veut_ecrire")`

```
f = open("my_file.txt", "w")
f.write("J'ecris dans mon fichier")
f.write("\n") #Retour a la ligne
f.write("C'est super genial")
f.write("\n")
f.write("mdr")
f.close()
```

[4] Le terme exact est *descripteur de fichier*

Lecture dans un fichier

On **ouvre** d'abord le fichier dans lequel on souhaite lire avec la permission de lecture. Et on stocke la valeur de retour de la fonction `open` dans une variable.
Lecture avec la fonction `read()` : variable_de_fichier.`read()` ou avec une boucle **for**.

```
f = open("my_file.txt", "r")
text = f.read()
print(text) #Imprime le contenu du fichier
f.close()
```

```
f = open("my_file.txt", "r")
for line in f:
    print(line) #Imprime toutes les lignes du fichier
f.close()
```

Fermeture d'un fichier

Une fois qu'on a fini de manipuler un fichier, on le ferme. C'est primordial ! Sinon on a une grande chance de générer une bonne grosse ribambelle d'erreurs.
Fermeture avec la fonction `close()` variable_de_fichier.`close()`

A vous de jouer !

Mission 0

Ecrivez votre prénom dans un fichier, EN UTILISANT PYTHON. Puis lisez le fichier, récupérez le prénom, et affichez "Bonjour [prénom]" dans la console.

A vous de jouer !

Mission 1

Réalisez une mini-base de donnée. Votre programme doit demander à l'utilisateur son nom, prénom, année de naissance, sexe, poids, viande préférée et carte préférée de hearthstone et retranscrire toutes ses informations dans le fichier "client1.txt" et ainsi de suite pour les autres clients.

1

Mission 2

Réalisez un programme pour aller recherche les informations de la mini base de donnée. Ce dernier demande à l'utilisateur de quel client il veut les information et lui imprime toutes les informations concernant ce client.

2

A vous de jouer !

Mission 3

Améliorez votre base de donnée pour qu'elle stocke implicitement (donc sans rien demander au client) l'espérance de vie du client et son âge.
L'espérance de vie d'un homme est de 72 ans et d'une femme de 80 ans.

Section 10

Fonctions

Quand l'utiliser?

Que faire si on a du code qui se répète régulièrement? Le copier-coller n'est pas considéré comme une bonne pratique, on préfère utiliser des **fonctions**, qui encapsulent une portion de code.
On essaye de rendre la fonction la plus générale possible, avec des **arguments**, des variables spéciales auxquelles la fonction a accès.

Exemple : tables de multiplication

```
# definition de la fonction
def tableDeMultiplication(numeroDeLaTable):
    i = 1
    while i <= 10:
        print(i*numeroDeLaTable)
        i += 1

# imprime la table de 4
tabledeMultiplication(4)

#imprime la table de 9, sans copier-coller!
tableDeMultiplication(9)
```

Définition générale

```
# Definition de la fonction
def nomDeLaFonction(argument1, argument2, ...):
    # corps de la fonction

# Appel de la fonction
nomDeLaFonction(arg1, arg2, ...)
```

Valeurs de retour

Jusqu'à présent, l'exemple donné montre une fonction qui travaille de manière "interne", elle ne renvoie aucune valeur. Mais c'est justement un des grands intérêts des fonctions!

```
def aireTriangle(largeur, hauteur):
    aire = largeur*hauteur
    return aire
    # ou plus rapide : return largeur*hauteur

print("L'aire est de " + str(aireTriangle(4, 9)))
```

Cet exemple montre une fonction qui calcule l'aire d'un rectangle, et l'utilisation de la valeur de retour.

A vous de jouer !

Missions

1. Créez les fonctions `soustraction(arg1, arg2)`, `multiplication(arg1, arg2)`, `division(arg1, arg2)`, et `power5(arg)` qui, comme leurs noms l'indique font respectivement les opérations de soustraction, multiplication, division, mise à la puissance de 5.
2. Créez la fonction stringLength(myString) qui affiche la longueur du texte en argument.

Portée des variables

Les variables sont déclarées dans leur "bloc". Une variable déclarée dans une fonction est distincte des autres variables déclarées en dehors du corps de la fonction, et n'est plus accessible quand la fonction se termine.

```
x = 10

def fonction():
    x = 20
    print(x)

fonction() # Imprime 20
print(x) # Imprime 10
```

Variables globales

On peut aussi utiliser utiliser le mot-clef `global` pour élargir la portée d'une variable, et la rendre globale.

```
x = 10
print(x) # Imprime 10

def fonction():
    global x
    x += 10
    print(x)

fonction() # Imprime 20
fonction() # Imprime 30
print(x) # Imprime 30
```

Dans ce cas, Python trouve `x` déclaré à un niveau supérieur, et la fonction y a pleinement accès et peut la modifier.

Section 11

Objets

Programmation orientée objet (POO)

La POO permet de définir ses propres types de données, sous forme d'objets. Chaque objet possède des variables (= d'autres objets) ainsi que des fonctions qui lui sont propres. Les variables sont appelées *attributs*, et les fonctions sont les *méthodes* de l'objet.

Par exemple, un objet String peut être défini comme suit :

- **attributs** : liste de caractères, longueur de la liste
- **méthodes** : `lower()`, `upper()`, ...

Classes et Objets

Il est important de bien faire la distinction entre une classe et un objet de cette classe.

- Une classe est une définition d'un concept. On peut la voir comme le mode de fabrication et d'utilisation d'un objet.
- Un objet est une instance d'une classe. Il s'agit d'un élément concret dont le comportement est défini par la classe.

Comme un objet effectue des opérations sur lui-même, il est important de pouvoir le référencer dans le code de la classe. On utilise pour ça le mot clef `self`.

Exemple de classe

```
class Animal:
    #Attributs
    #Valeurs par defaut
    nombrePattes = 4

    #Constructeur
    #Permet de creer un objet animal avec un certain nom
    def __init__(self, nom):
        self.nom = nom

    #Modificateur
    #casse une patte a l'animal
    def patte_cassee(self, couleur):
        self.nombrePattes -= 1
```