

Découverte de l'informatique *#2*

Matthieu Nicolas
Polytech S1

Plan

- Débrief TD1
- Notions pour TD2
- TD2

Débrief TD1

Découverte de l'informatique
#2

print() - 1

- **Fonction** qui permet d'**afficher** une **chaîne de caractères** à l'utilisateur

```
>>> print("hello world")  
hello world  
>>> █
```

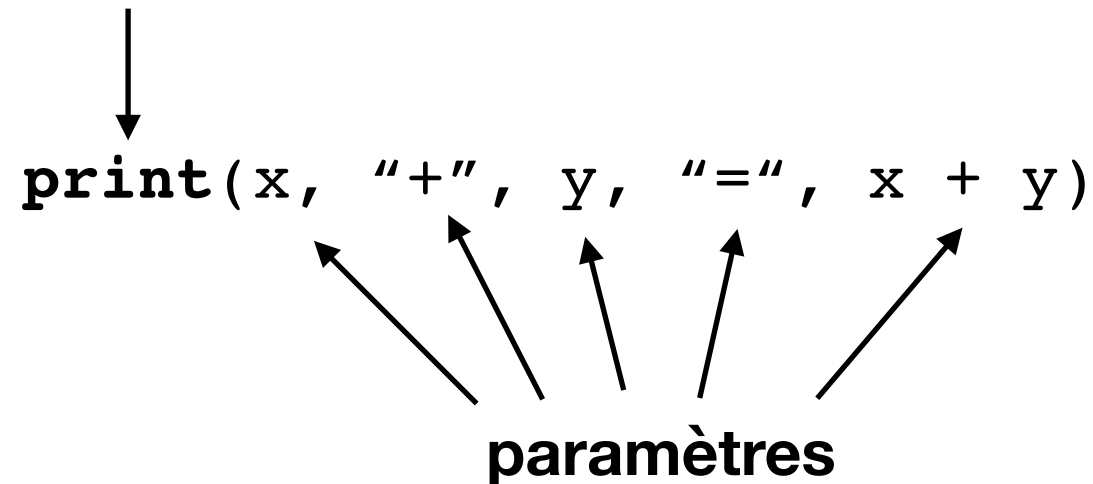
print() - 2

- Permet aussi d'**afficher** des **variables** à l'utilisateur

```
>>> x = 5
>>> y = 7
>>> print(x, "+", y, "=", x + y)
5 + 7 = 12
>>> █
```

Appel de fonction

nom de la fonction



`print(x, "+", y, "=", x + y)`

paramètres

The diagram shows the function call `print(x, "+", y, "=", x + y)`. An arrow points from the text 'nom de la fonction' to the word 'print'. Five arrows point from the text 'paramètres' to the arguments: 'x', '+', 'y', '=', and 'x + y'.

- **Données** utilisées par la fonction **pour produire** son **résultat**
- **Passés** à la fonction en les listant entre les parenthèses de l'appel de fonction
 - Séparés par des ,

Paramètres

- Une fonction peut prendre **0**, **1**, **n** ou une **infinité** de paramètres
 - Dépend de la fonction
- L'**ordre** des paramètres est **important**
 - Le **rôle** d'un paramètre **dépend** de sa **position**

```
>>> print("hello", "world")
hello world
>>> █
```

```
>>> print("world", "hello")
world hello
>>> █
```

Paramètres nommés

- Une **fonction** peut avoir des **paramètres nommés** (*keyword parameters*)

```
>>> print("hello", "world", sep="$$$")
hello$$$world
>>> █
```

- **print()** possède, entre autres :
 - **sep**, qui permet de **définir le séparateur** affiché **entre** les différents **paramètres** (" " par défaut)
 - **end**, qui permet de **définir le motif ajouté** à la fin de la **chaîne de caractères** ("\n" par défaut)

Boucle For - 1

- Permet de **répéter** un **ensemble d'instructions** un **nombre défini de fois**

```
>>> for i in range(5):  
...     print("hello", "world")  
...  
hello world  
hello world  
hello world  
hello world  
hello world  
>>> █
```

Boucle For - 2

compteur/variable d'itération



```
for i in range(5):
```



nombre d'itérations

- **range()** permet de définir le nombre de fois que l'on va boucler (nombre d'**itérations**)
- La variable **i** va changer de valeur au fur et à mesure de l'itération en cours

Boucle For - 3

```
>>> for i in range(5):  
...     print(i)  
...  
0  
1  
2  
3  
4  
>>>
```

- On remarque que **i varie de 0 à 4**
- **i est incrémenté à la fin de chaque itération**
- Une fois que **i atteint la valeur finale (5), on s'arrête**

range()

- **range()** permet de définir le nombre d'itérations
- Plus précisément, **définit les valeurs que i va prendre** au cours des itérations successives
- Son **comportement change** en fonction du **nombre de paramètres passés**

range(n)

```
>>> for i in range(5):  
...     print(i)  
...  
0  
1  
2  
3  
4  
>>> 
```

- **i** varie de **0** à **n**, **n exclu**
- **i augmente de 1** à chaque itération

range(m, n)

```
>>> for i in range(1, 5):  
...     print(i)  
...  
1  
2  
3  
4  
>>> █
```

- **i** varie de **m** à **n**, **n exclu**
- **i augmente de 1** à chaque itération

range(m, n, p)

```
>>> for i in range(1, 5, 2):  
...     print(i)  
...  
1  
3  
>>> █
```

- **i** varie de **m** à **n**, **n exclu**
- **i** augmente de **p** à chaque itération

Notions pour TD2

Découverte de l'informatique
#2

Objectifs

- Vous faire découvrir **randint()**
- Vous faire utiliser les **fonctions mathématiques** de base
- Aborder les **instructions conditionnelles**

randint(x, y)

- Permet de **générer** un **nombre aléatoire entre x et y**, x et y inclus

```
>>> randint(1, 9999)
3289
>>> █
```

Modules et import - 1

- Petit problème : **randint()** n'est pas disponible dans Python de base
- L'idée est d'**éviter de surcharger Python** avec TOUTES les fonctions existantes
 - Mais **charger initialement** que les **fonctions principales**
 - Et laisser le soin **aux devs d'importer** les **fonctions** dont ils ont **besoin**

Modules et import - 2

- Nécessaire d'**importer la fonction** au préalable
- Pour cela, utilise l'instruction suivante :

```
>>> from random import randint  
>>> █
```

- où **random** est le module
- **randint** la fonction importée

Modules et import - 3

- Peut **importer tout le module** d'un coup
- Peut alors **accéder à ses fonctions et variables** de la manière suivante :

```
>>> import math
>>> math.pi
3.141592653589793
>>> █
```

- Peut accéder à la documentation d'un module après l'avoir chargé grâce à **help()**

Fonctions mathématiques

- Regroupées dans le module **math**
- Dispose de **quelques fonctions de base** sans ce module :
 - **abs()** permet d'obtenir la valeur absolue d'un nombre
 - **round()** permet d'arrondir un nombre à la décimale souhaitée

Instructions conditionnelles

- 1

- Permet d'**effectuer des instructions seulement si une condition est vérifiée** (*condition == True*)
- Permet de **différencier les différents cas** à traiter
- S'utilise grâce à l'opérateur **if**

Instructions conditionnelles

- 2

```
x = eval(input("Saisir un nombre :"))
y = eval(input("Saisir un autre nombre :"))
if (x > y):
    print(x, "est plus grand que", y)
```

- Que fait ce code ?
 - **Compare** 2 nombres **x** et **y**
 - **Affiche** un **message** si **x** est le **plus grand** des 2

Instructions conditionnelles

- 3

- Peut vouloir **déclencher des instructions** dans le cas où la **condition n'est pas vérifiée** (*condition == False*)
- Se fait grâce à l'opérateur **else**

```
x = eval(input("Saisir un nombre :"))
y = eval(input("Saisir un autre nombre :"))
if (x > y):
    print(x, "est plus grand que", y)
else:
    print(y, "est plus grand que", x)
```

Instructions conditionnelles

- 4

- Peut vouloir **tester différents cas** à la suite
- Se fait grâce à l'opérateur **elif**

```
x = eval(input("Saisir un nombre :"))
y = eval(input("Saisir un autre nombre :"))
if (x > y):
    print(x, "est plus grand que", y)
elif (y > x):
    print(y, "est plus grand que", x)
else:
    print(x, "est égal à", y)
```

Remarques

- Possible d'**ajouter autant de elif** que nécessaire
- **Exécute au maximum un des branchements** du *if ... elif ... elif... else ...*
- Les branchements suivants seront **ignorés**, même si *condition == True*

Conditions booléennes - 1

- **Expression logique** qui **compare** des **variables** et/ou **valeurs** entre elles à l'aide d'**opérateurs** (`==`, `!=`, `<`, `<=`, `>`, `>=`)
 - `x == 5`
 - `x < y`
 - `"anaconda" < "python"`
- Renvoie **True** si vraie, **False** sinon

Conditions booléennes - 2

- Peut construire des **conditions plus complexes** à l'aide des opérateurs **and** (et), **or** (ou) et **not** (non)
 - $0 < x \text{ and } x < 100$

Tables de vérité

x and y

x / y	FALSE	TRUE
FALSE	FALSE	FALSE
TRUE	FALSE	TRUE

x or y

x / y	FALSE	TRUE
FALSE	FALSE	TRUE
TRUE	TRUE	TRUE

not x

x	FALSE	TRUE
not x	TRUE	FALSE

TD2

Découverte de l'informatique
#2

TD1 - Boucle For

- Si ce n'est pas déjà fait, compléter **exercices 6 et 9** du **chapitre 2, page 15**

TD2 - Nombres et Mathématiques

- **Chapitre 3, page 23**
 - Exercices 1, 3, 4, 6, 8, 13, 15

TD2 - Instructions conditionnelles

- **Chapitre 4, page 30**
 - Exercices 1, 3, 5, 9, 12

Avez-vous des questions ?