

# ICC: Programmation

GC/MX, Cours 2, 27 septembre 2019

Jean-Philippe Pellet

# Previously, on Programmation...

---

- Nous utilisons **VS Code** pour programmer en **Python**
- Quelques **types** de base en Python: **int**, **float**, **str**
- **Conversion** entre ces types
- **Déclaration d'une variable** avec valeur initiale (*type optionnel*):

`<nomDeVariable>: <type> = <valeur>`

```
age: int = 36
height: float = 1.79
my_name: str = "Jean-Philippe"
```

- **Méthodes, fonctions et slicing** pour calculer des valeurs dérivées:

```
upper_name: str = my_name.upper()
name_length: int = len(my_name)
first_part: str = my_name[0:4]
```

- **Google et exemples sur le net** pour rechercher comment faire quelque chose

# Appris aux exercices: Méthodes et fonctions

---

- Méthode

➔ `upper_name: str = my_name.upper()`

- S'écrit **après** le nom de la variable suivi d'un point
- Toujours avec parenthèses
- Dépend du type de la variable

- Fonction

➔ `name_length: int = len(my_name)`

- S'écrit **sans préfixe** avec arguments entre parenthèses
- Applicable en général à plusieurs types de données
- Nombre total de fonctions restreint

# Appris aux exercices: Slicing

- Slicing

➔ `first_part: str = my_name[0:4]`

— Forme simple: deux indices entre crochet après nom de variable

— S'applique aux strings (*et listes, tuples, etc.*)

— Premier indice optionnel si égal à 0

➔ `first_part: str = my_name[:4]`

— Second indice optionnel si égal à `len(variable)`

➔ `second_part: str = my_name[5:len(my_name)]`  
`second_part: str = my_name[5:] # équivalent`

— *Troisième indice possible... On en reparle*

# Appris aux exercices: f-Strings

---

- Un **f-string** est un string précédé de f

➔ 

```
print(f"Durée du trajet: {duration} h")
```

— Les expressions entre accolades `{}` sont **évaluées comme code** Python et insérées dans le string final

— Plus pratique que de devoir concaténer des strings en convertissant les valeurs non-strings:

➔ 

```
print("Durée du trajet: " + str(duration) + " h")
```

— Pour insérer une accolade dans un f-string: on la double

➔ 

```
print(f"On a l'ensemble A = {{ x | x > {min_value} }}")
```

# Appris aux exercices: Bits and pieces

---

- Deux opérateurs de division
  - $a / b$ : division **normale**, renvoie un float
  - $a // b$ : division **euclidienne**, renvoie un int (*ou float arrondi si  $a$  ou  $b$  est un float*)
  - $a \% b$ : **modulo**, reste de la division euclidienne
- Une fonction/méthode peut **renvoyer plusieurs valeurs** (*en fait, un tuple*)
  - ➔ 

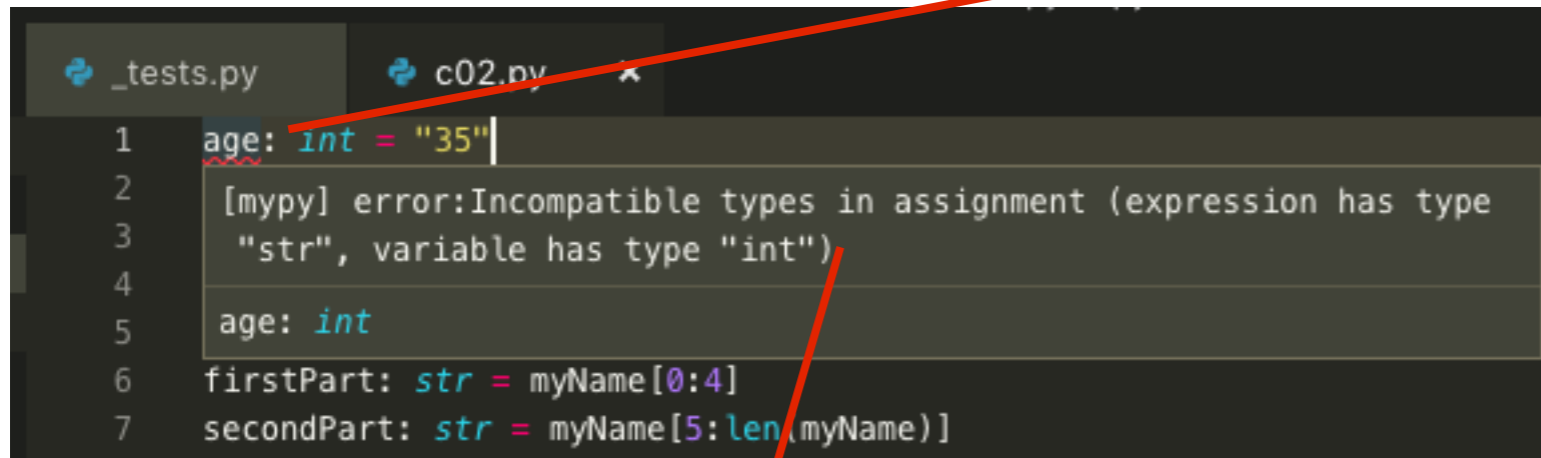
```
duration_hours, rest = divmod(distance, speed)
```
  - *On en reparle*

# Les premières semaines...



# Erreurs du compilateur/linter

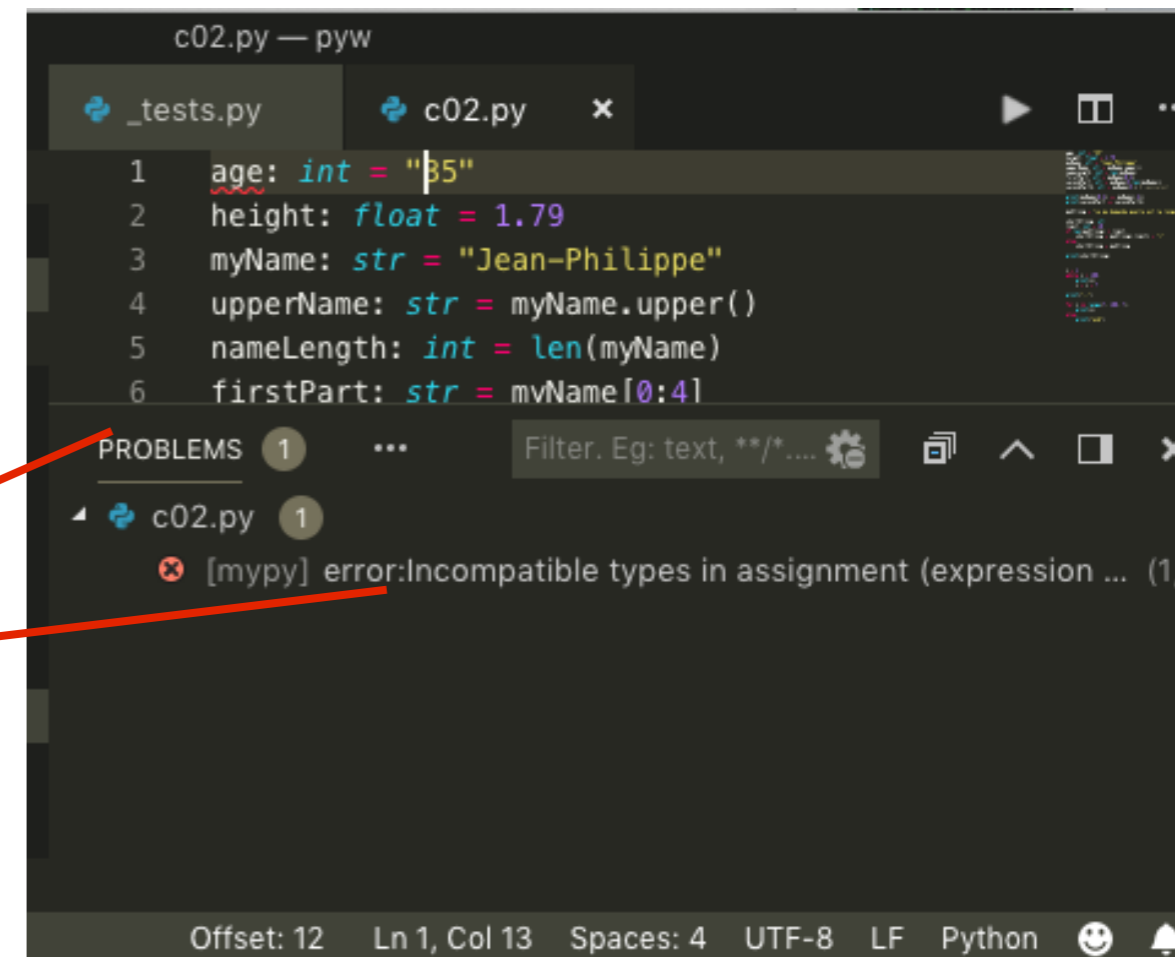
Les erreurs sont signalées **en rouge**.  
Résolvez-les avant de faire tourner  
votre code.



```
1 age: int = "35"  
2  
3  
4  
5 age: int  
6 firstPart: str = myName[0:4]  
7 secondPart: str = myName[5:len(myName)]
```

Déplacez votre curseur au-dessus de l'erreur  
pour voir une explication...

... ou affichez l'onglet Problems en  
bas de la fenêtre



```
c02.py — pyw  
1 age: int = "35"  
2 height: float = 1.79  
3 myName: str = "Jean-Philippe"  
4 upperName: str = myName.upper()  
5 nameLength: int = len(myName)  
6 firstPart: str = mvName[0:4]
```

PROBLEMS 1 ... Filter: Eg: text, \*\*/\*...  
c02.py 1  
[mypy] error:Incompatible types in assignment (expression ... (1)

Offset: 12 Ln 1, Col 13 Spaces: 4 UTF-8 LF Python



# Style du code

---

*Conventions seulement, mais utile pour s'y retrouver. **Suivez-les!***

- **Nom des variables et méthodes**
  - Pas d'espaces! Pas de caractères spéciaux ou accentués
  - Pas de majuscules
  - Des underscores pour séparer les mots
  - Exemples: `age`, `my_name`, `first_name`
- **Indentation**
  - Nombre d'espaces ou tabs avant le début de la ligne
  - Change la signification du code... *On en reparlera*

# Problèmes communs

---

Je n'arrive pas à faire tourner mon code correctement, mais le même fichier marchait avant

Problème: vous avez ouvert un fichier et pas le workspace de VS Code

→ *Faites Open Workspace dans VS Code et choisissez le fichier `__workspace__.code-workspace` dans le dossier `myfiles/Programmation/icc`*

Je ne retrouve pas mes fichiers de la séance passée

Problème: votre workspace n'est pas dans *myfiles*

→ *Relisez les consignes de la séance 1 et demandez à un assistant*

Je ne vois pas le dossier *myfiles* sur le bureau

Problème potentiel: vous vous êtes identifiés en utilisant des majuscules dans votre nom d'utilisateur

→ *Faites une demande à [1234@epfl.ch](mailto:1234@epfl.ch) (et moi en CC) en décrivant le problème*

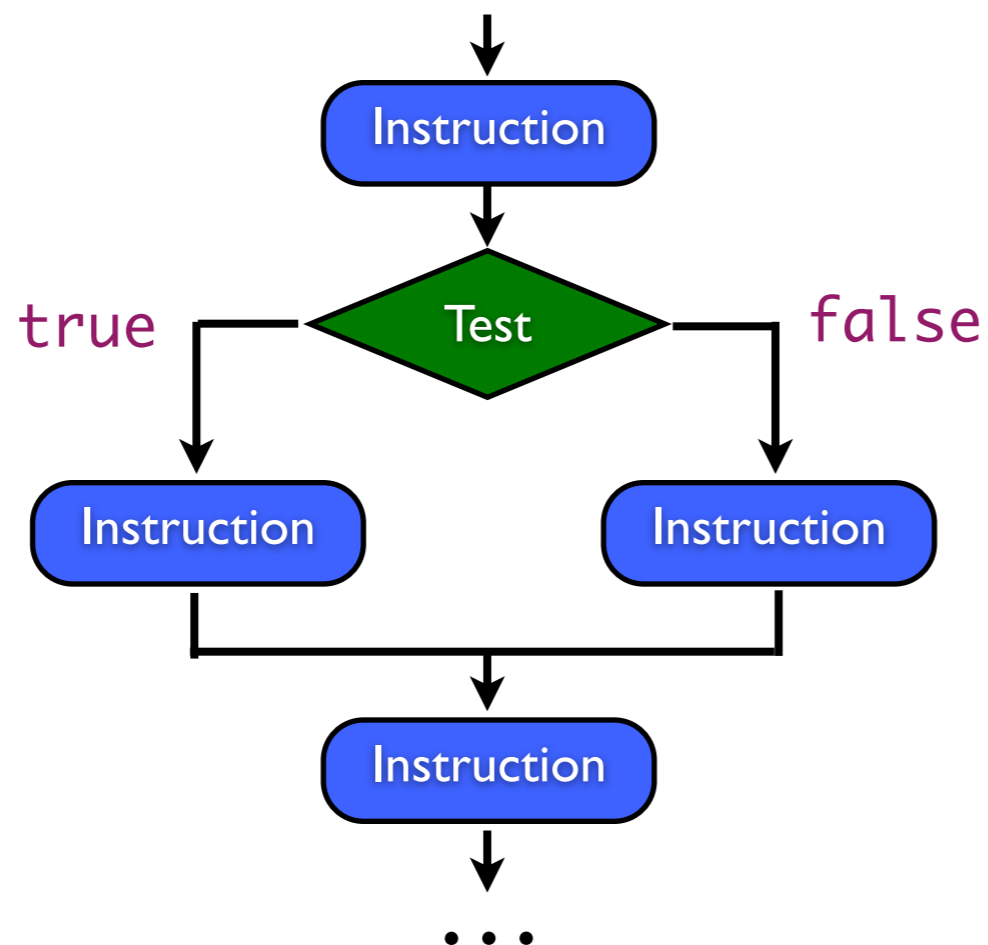
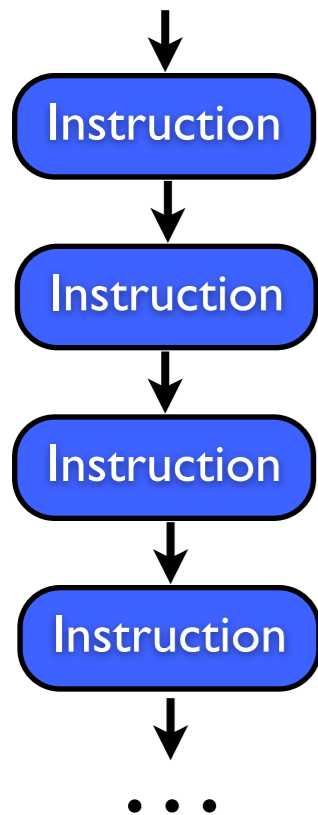
# Cours de cette semaine

*Conditions*

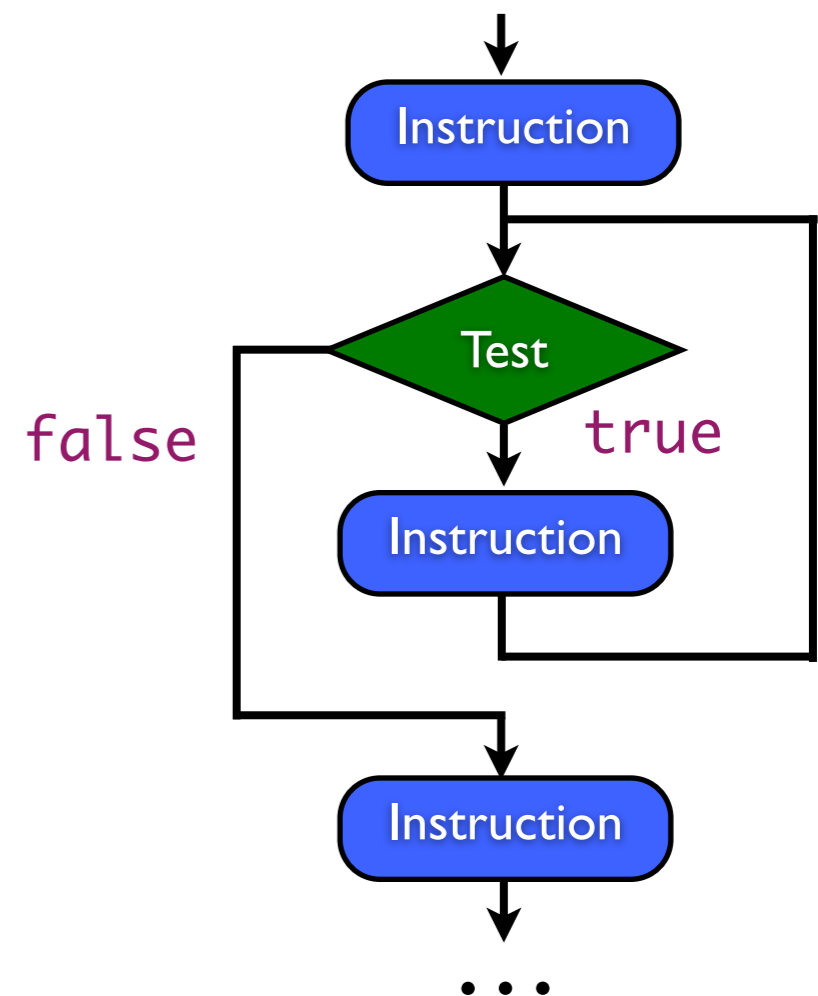
*Boucles*

# Motivation

- **Conditions:**  
«Si ce string est plus long que 20 caractères, raccourcis-le et rajoute “...” à la fin»

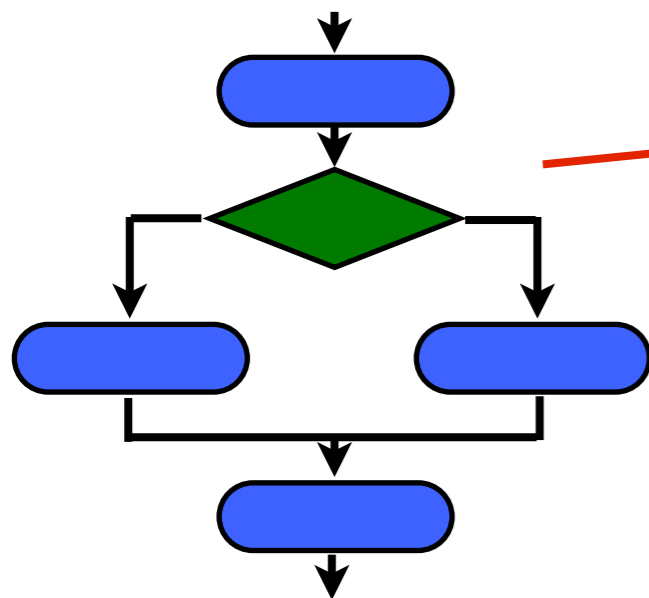


- **Boucles:**  
«Affiche tous les multiples de 7 plus petits que 100.»



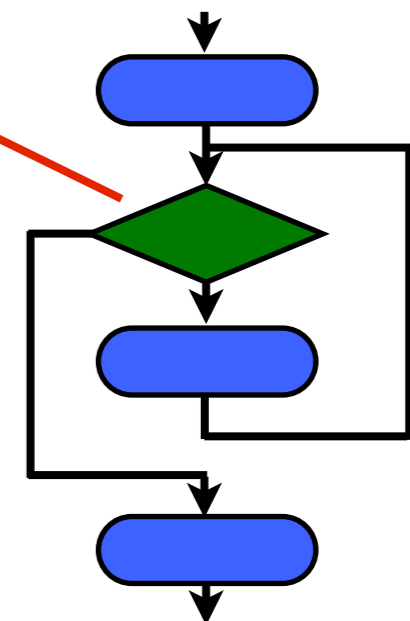
# Conditions et boucles

- Des éléments de base de la programmation, appelées *Structures de contrôle* (*control flow*).
- Type central: `bool` (valeur booléenne)
  - soit vrai (`True`), soit faux (`False`)
- Chaque test, chaque condition a besoin d'un `bool` pour savoir comment continuer



Boolean testé  
une fois

Boolean testé  
lors de chaque  
passage



# Conditions: *If*

---

«Si ce string est plus long que 20 caractères, raccourcis-le et rajoute “...” à la fin»

*Démo*

# Notre exemple ligne par ligne

```
my_string: str = "je me demande quelle est la longueur de ceci"
limit: int = 10

if len(my_string) > limit:
    short_string = my_string[:limit - 1] + "..."
else:
    short_string = my_string

print(short_string)
```

**Condition** qui produit un bool, suivie d'un deux-points

Suite normale du programme  
(code non indenté)

Que faire si la condition est **fausse**  
(code *indenté*)

Que faire si la condition est **vraie**  
(code *indenté*)

# Forme générale d'un *If*

---

```
if <bool_condition>:  
    # code if bool_condition is True
```

---

```
if <bool_condition>:  
    # code if bool_condition is True  
else:  
    # code if bool_condition is False
```

---

```
if <bool_condition1>:  
    # code if bool_condition1 is True  
elif <bool_condition2>:  
    # code if bool_condition1 is False and  
    # bool_condition2 is True  
else:  
    # code if both conditions are False
```



# Comment obtenir des booleans?

---

## *Avec des ints*

```
age: int = 19
if age < 12: # plus petit
if age > 18: # plus grand
if age <= 18: # plus petit ou égal
if age >= 18: # plus grand ou égal
if age == 50: # égal
if age != 13: # non égal
```

## *Avec des floats*

```
price: float = 1.2
if price < 2.5: # plus petit
if price > 1.3: # plus grand
# égalité et inégalité possibles, mais
# attention aux imprécisions en virgule flottante
```

# Comment obtenir des booléens?

## Avec des strings

```
course: str = "Programmation"
if course == "Programmation":           # égalité
if course.lower() == "programmation":   # sans tester les majuscules
if course.startswith("Prog"):          # test de préfixe
if course.endswith("ation"):           # test de suffixe
if "mmati" in course:                  # test de contenance
```

## Avec des booléens

```
x: int = ...
condition1: bool = x > 1    # une condition stockée dans une variable
condition2: bool = x < 100  # une autre

if condition1 and condition2: # conjonction: vrai si les deux sont vraies
if condition1 or condition2:  # disjonction: vrai si l'une des deux est vraie
if not condition1:           # inversion: vrai devient faux et inversement
if (not condition1) or (condition1 and condition2): # combinaisons...
```

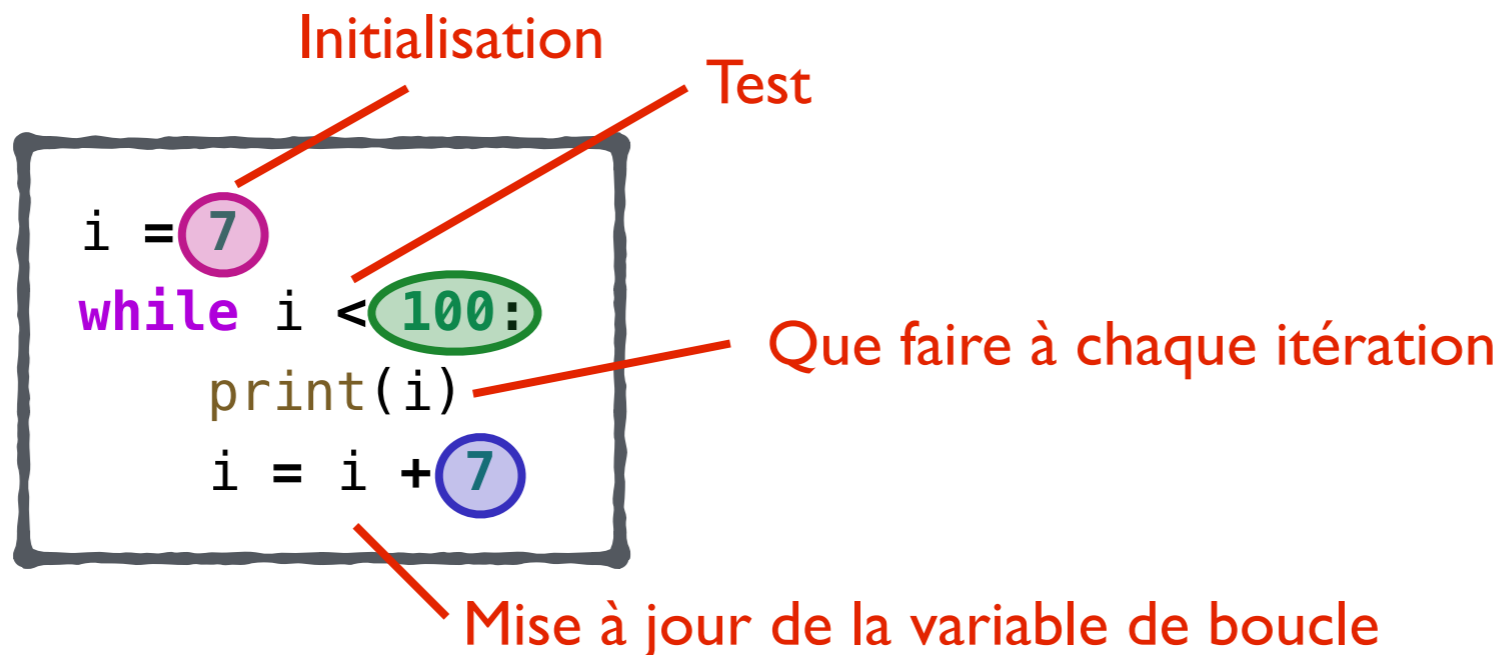
# Boucles: *While* et *For*

---

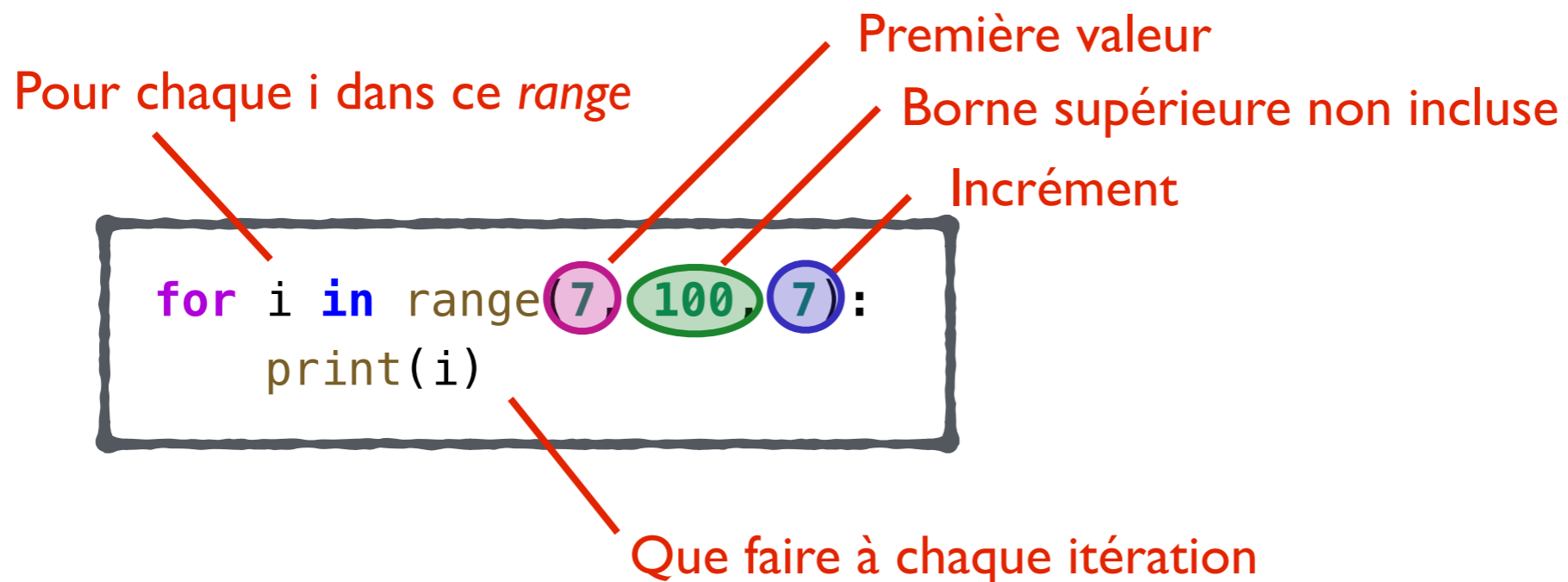
*« Affiche tous les multiples de 7 plus petits que 100 »*

*Démo*

# Notre exemple ligne par ligne



**Début**  
**Changement**  
**Fin**



# La fonction *range()*

- Très utile pour le *for-in*
  - 1 argument: `range(y)` → de 0 (inclus) à `y` (exclu)
  - 2 arguments: `range(x, y)` → de `x` (inclus) à `y` (exclu)
  - 3 arguments: `range(x, y, s)` → idem en ajoutant `s`
- Exemples:

```
range(10) # on part de 0 et on s'arrête avant 10, donc 9
range(0, 10) # même chose
range(2, 10) # on commence à 2 plutôt qu'à 0
range(2, 10, 3) # on incrémente de 3 plutôt que de 1
range(100, 0, -10) # 100, 90, 80, ..., 20, 10
```

- Dans l'interpréteur:

```
list(range(...))
```

Montrera une liste avec toutes  
les valeurs de la *range*

# Résumé Cours 2

---

- Les **conditions** et les **boucles** sont des structures de contrôles vitales en programmation
- Les **booléens** sont à la base des décisions qu'on prend avec *if* ou *while*
- On obtient ces booléens avec, par exemple, des **comparaisons** (e.g. sur des types numériques) ou en **appelant des méthodes** (e.g. sur un string)
- Les booléens peuvent se **combiner logiquement** entre eux avec les opérateurs `and`, `or` et `not`
- La fonction ***range()*** permet de faire des boucles plus simplement avec *for-in*