EPFL

ICC

Programmation

CGC/SIE, **Cours 2**

26 février 2024

Jean-Philippe Pellet

Previously, on ICC 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*):

```
age = 41
height = 1.80
my_name = "Jean-Philippe"
```

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

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

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

Programme du cours

8 avril

Examen intermédiaire programmation

15%

Session d'examen

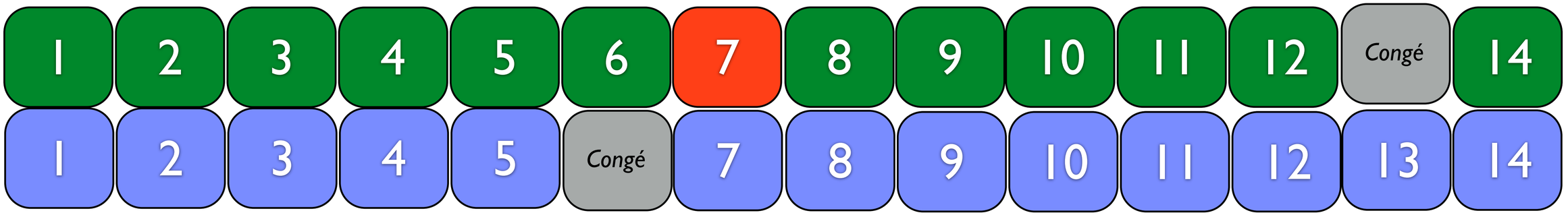
Examen final

75%

Cours & séries, partie programmation

10%

*Cours, séries & **mini**projet, partie programmation*



Cours & séries, partie théorique

Cours & séries, partie théorique

*L'examen final porte sur la partie théorique (⅔) **et** la partie programmation (⅓)*

Appris aux exercices: Méthodes et fonctions

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

```
upper_name = my_name.upper()
```

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

```
name_length = len(my_name)
```

Appris aux exercices: Slicing

- Slicing

```
first_part = 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 = my_name[:4]
```

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

```
second_part = my_name[5:len(my_name)]
second_part = 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

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

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

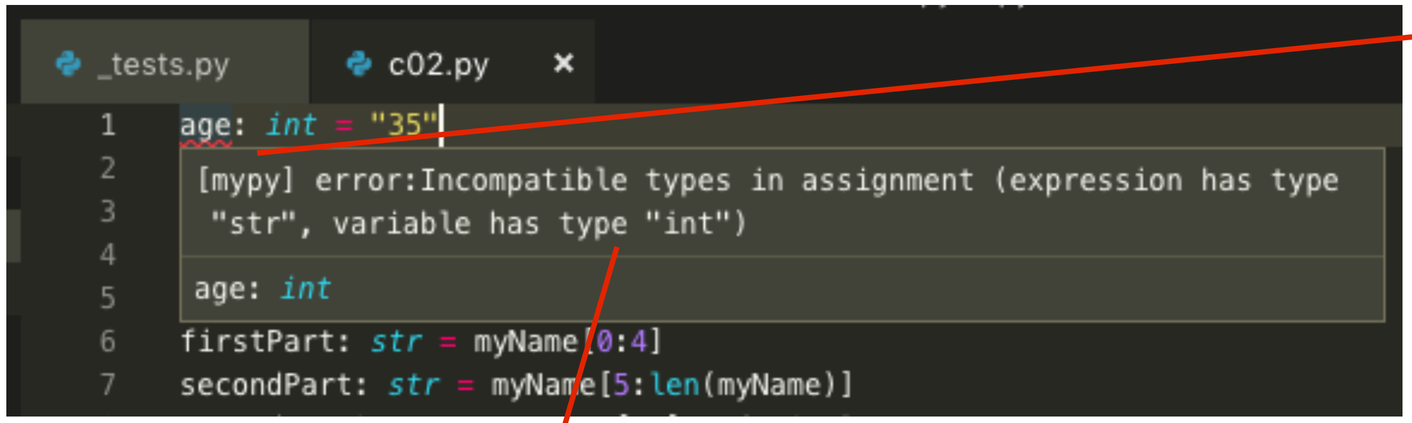
Appris aux exercices: Bits and pieces

- 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 reparlera*

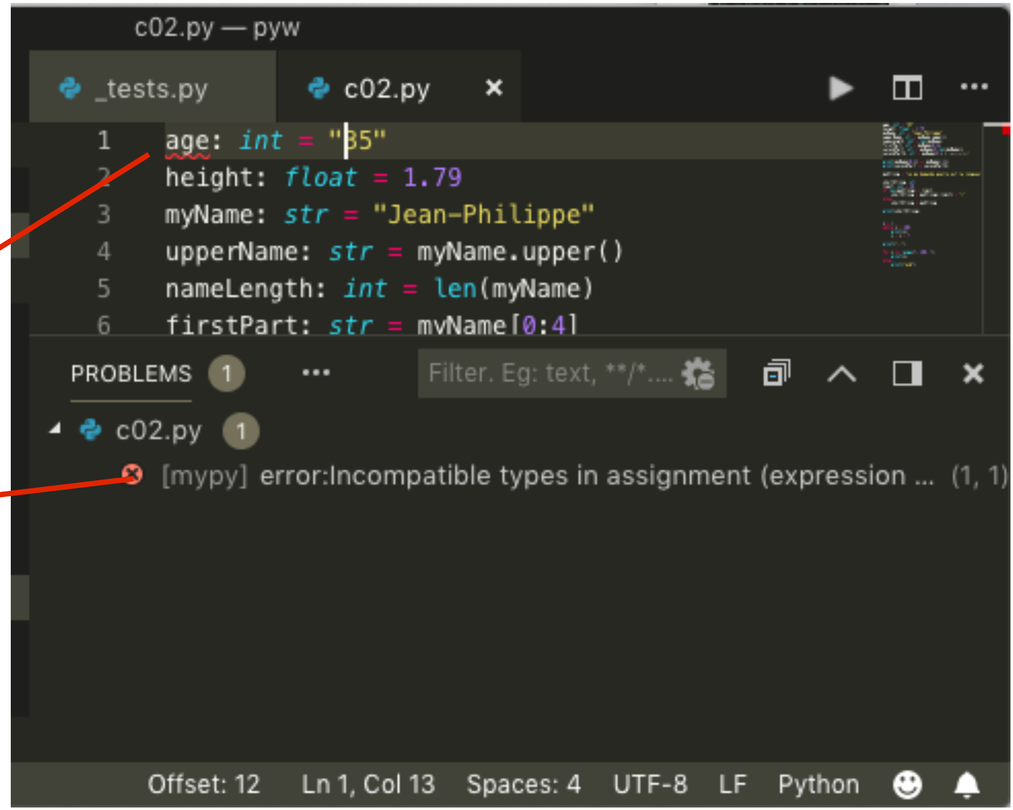
Erreurs du compilateur/linter



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

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

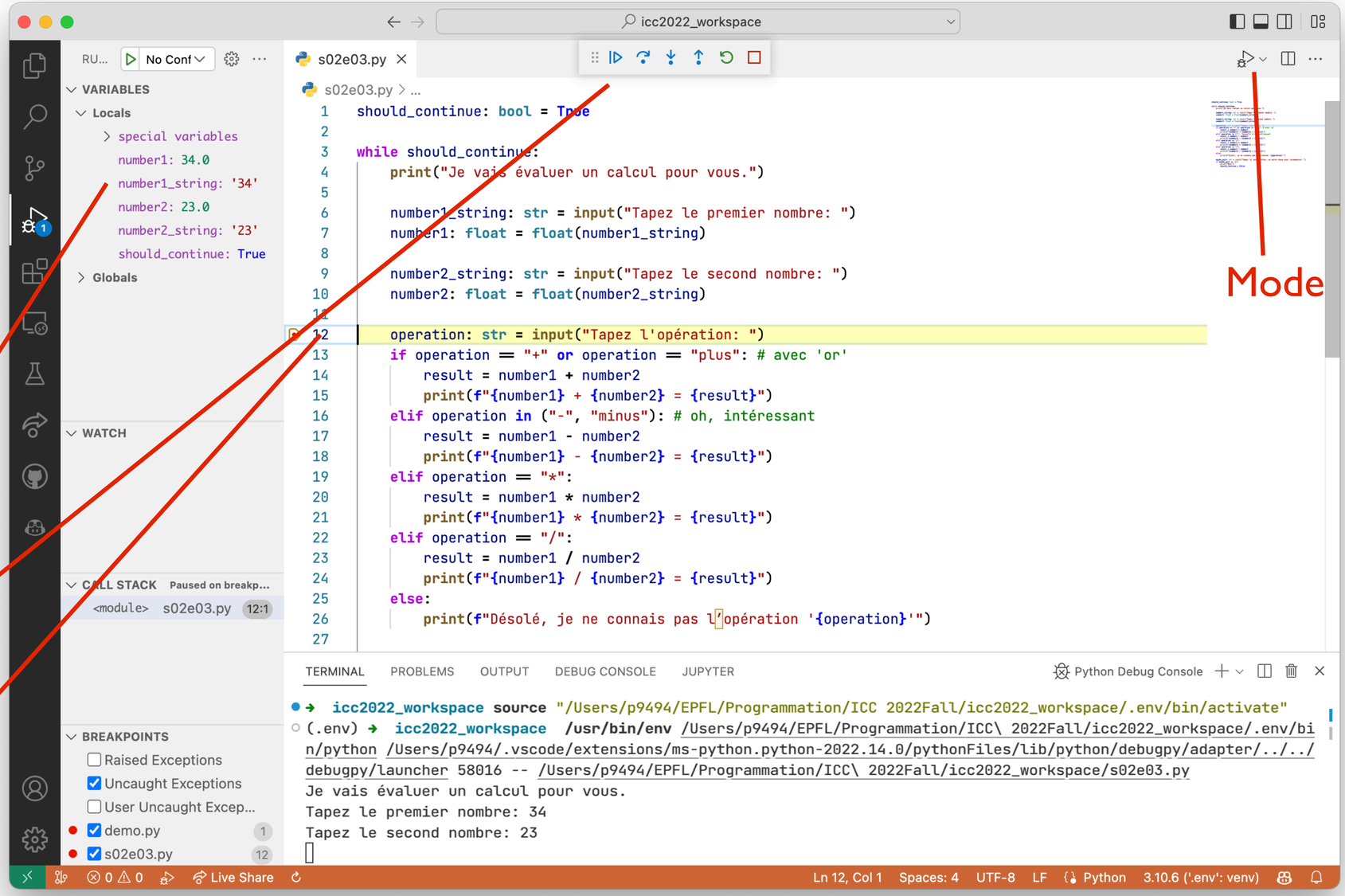
… ou affichez l'onglet Problems *en bas de la fenêtre*



Debugger

Démo

Un programme démarré en mode Debug s'arrête au premier breakpoint trouvé et permet d'avancer pas à pas



Style du code

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

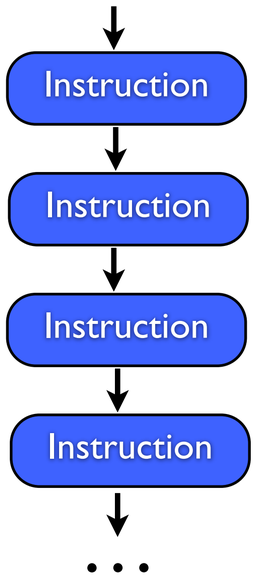
- Nom des variables et méthodes
  - Pas d'espaces! Pas de caractères spéciaux ou accentués
  - Pas de majuscules
  - Des underscores pour séparer les mots
  - Exemples: `age`, `my_name`, `first_name`
- Indentation
  - Nombre d'espaces ou tabs avant le début de la ligne
  - Change la signification du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

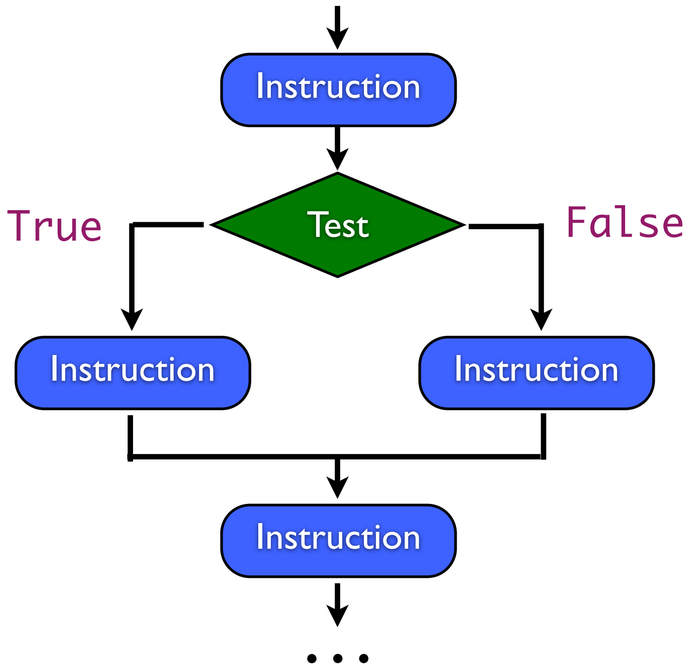
Cours de cette semaine

Conditions
Boucles

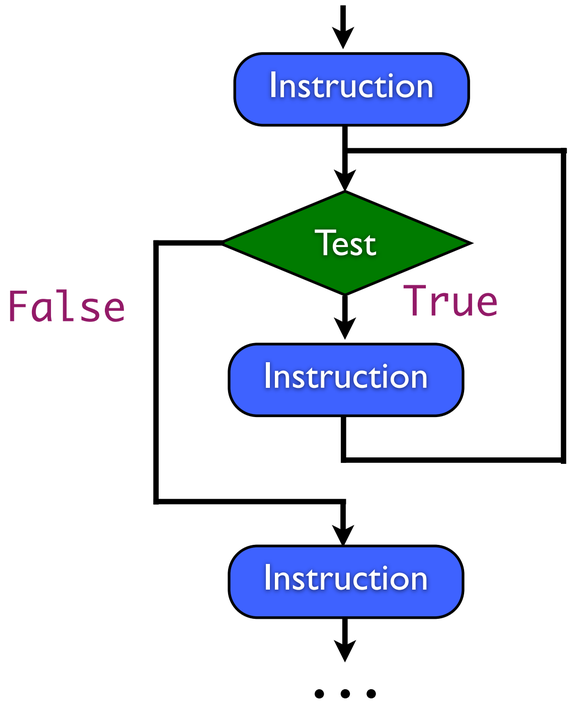
Motivation

- Conditions:
  «Si ce string est plus long que 20 caractères, raccourcis-le et rajoute "…" à la fin»



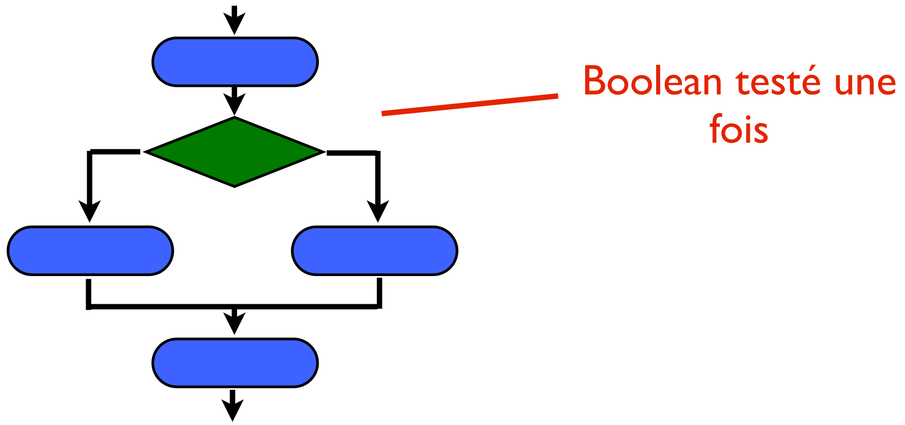


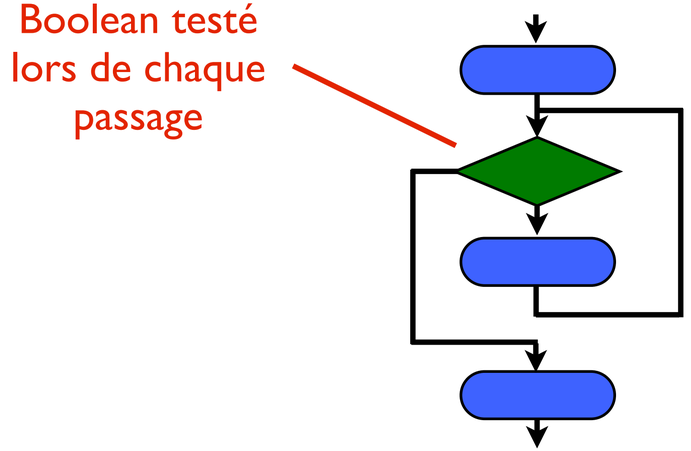
- Boucles:
  «Affiche tous les multiples de 7 plus petits que 100.»



Conditions et boucles

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





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 = "je me demande quelle est la longueur de ceci"
limit = 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

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

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

Suite normale du programme (code non 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 = 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 = 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 booleans?

Avec des strings

```
course = "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

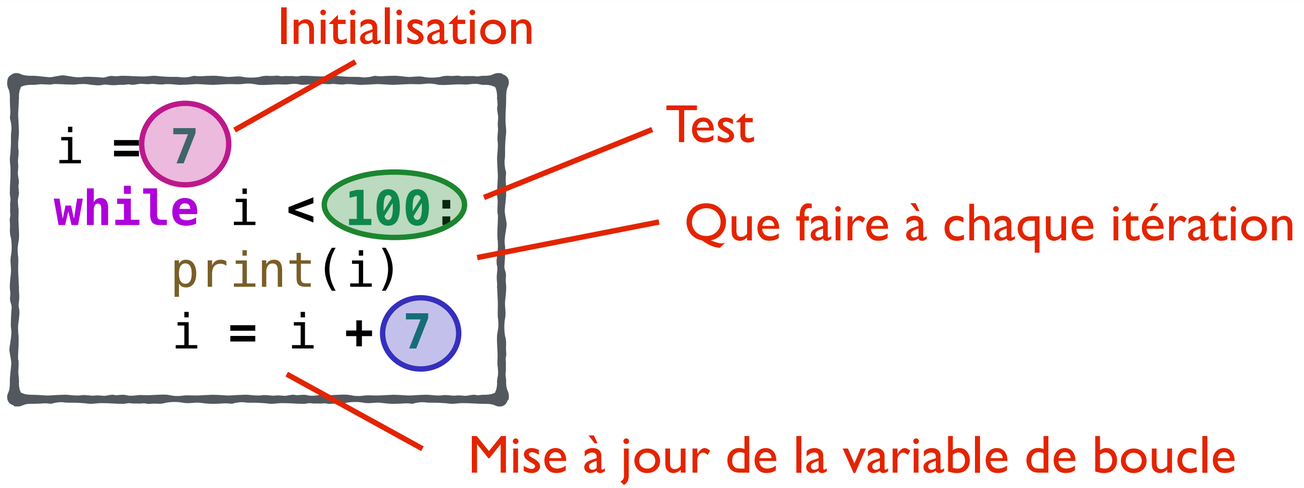
```
x: int = ...

if x > 1 and x < 100: # conjonction: vrai si les deux sont vraies
if x > 1 or x < 100:  # disjonction: vrai si l'une des deux est vraie
if not (x > 1):             # inversion: vrai devient faux et inversement
if (not x > 1) or (x > 1 and x < 100): # combinaisons...
```

Boucles: *While* et *For*

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

Démo

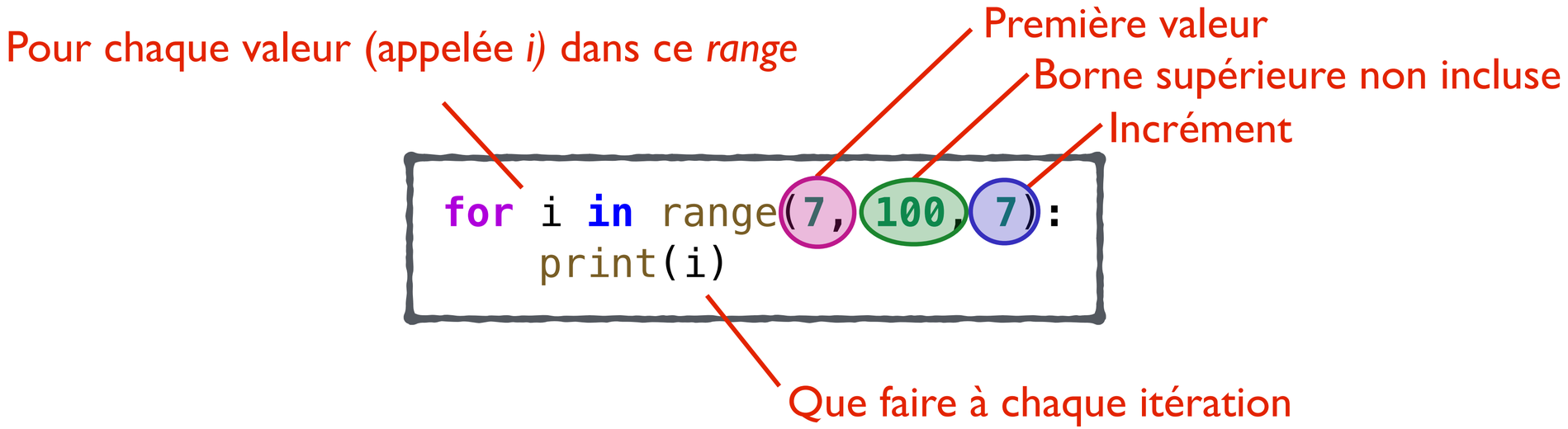
Notre exemple ligne par ligne



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 essentielles en programmation
- Les booléens sont à la base des décisions qu'on prend avec *if* ou *while*
- On obtient ces booléens avec, entre autres, des comparaisons (p. ex. sur des types numériques) ou en appelant des méthodes (p. ex. 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*