

Programmation

CGC/SIE, Cours 2

26 février 2024

Jean-Philippe Pellet

```

class ProgramView(Canvas):
    def __init__(self, parent, *args) -> None:
        Canvas.__init__(self, parent, height=2 * TOP_MARGIN + 4 * LINE_HEIGHT, highlightthickness=0, *args)

    def redraw(
        self,
        program: Program,
        current_subprogram: List[Instruction],
        current_instruction_index: int,
    ) -> None:
        height = self.winfo_height()
        width = self.winfo_width()

        self.delete(ALL)
        self.create_rectangle(0, 0, width, height, fill=window_background_color, width=0)

        # boucle pour les 4 sous-programmes P1 à P4
        for i, subprogram in enumerate(
            [program.P1, program.P2, program.P3, program.P4]
        ):
            # dessin du titre
            instruction_center_y = TOP_MARGIN + 1 * LINE_HEIGHT + LINE_HEIGHT // 2
            self.create_text(LEFT_MARGIN // 2, instruction_center_y, text=f"P{i + 1}")

            # dessin de chaque instruction
            for j, instr in enumerate(subprogram):
                instruction_center_x = (
                    LEFT_MARGIN
                    + j * (INSTRUCTION_BOX_SPACING + INSTRUCTION_BOX_WIDTH)
                    + INSTRUCTION_BOX_WIDTH // 2
                )
                instruction_x = instruction_center_x - INSTRUCTION_BOX_WIDTH // 2
                instruction_y = instruction_center_y - INSTRUCTION_BOX_HEIGHT // 2
                instruction_width = INSTRUCTION_BOX_WIDTH
                instruction_height = INSTRUCTION_BOX_HEIGHT

```

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"
```

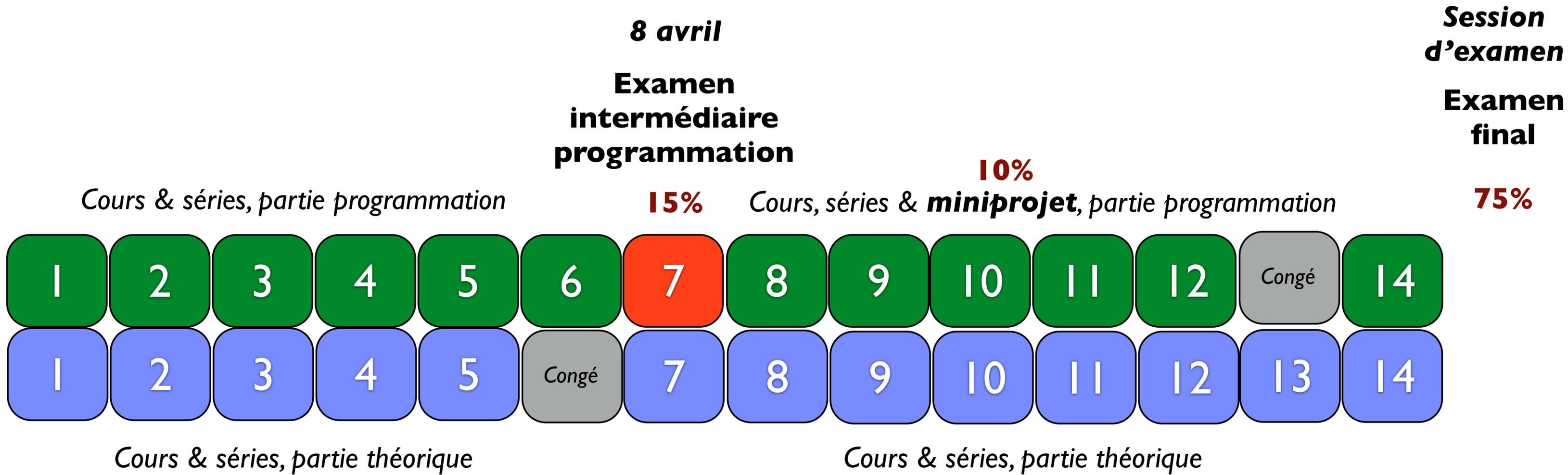
- **Méthodes, fonctions et slicing** pour calculer des valeurs dérivées:

```
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



*L'examen final porte sur la partie théorique (2/3) **et** la partie programmation (1/3)*

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

```
_tests.py  c02.py  x
1  age: int = "35"
2  [mypy] error:Incompatible types in assignment (expression has type
3  "str", variable has type "int")
4
5  age: int
6  firstPart: str = myName[0:4]
7  secondPart: str = myName[5:len(myName)]
```

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...

```
c02.py — pyw
_tests.py  c02.py  x
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, 1)

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

... 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

État des variables

Outils pour continuer l'exécution

Breakpoint

The screenshot shows a Python IDE in debug mode. The main window displays a Python script named `s02e03.py` with the following code:

```
1  should_continue: bool = True
2
3  while should_continue:
4      print("Je vais évaluer un calcul pour vous.")
5
6      number1_string: str = input("Tapez le premier nombre: ")
7      number1: float = float(number1_string)
8
9      number2_string: str = input("Tapez le second nombre: ")
10     number2: float = float(number2_string)
11
12     operation: str = input("Tapez l'opération: ")
13     if operation == "+" or operation == "plus": # avec 'or'
14         result = number1 + number2
15         print(f"{number1} + {number2} = {result}")
16     elif operation in ("-", "moins"): # oh, intéressant
17         result = number1 - number2
18         print(f"{number1} - {number2} = {result}")
19     elif operation == "*":
20         result = number1 * number2
21         print(f"{number1} * {number2} = {result}")
22     elif operation == "/":
23         result = number1 / number2
24         print(f"{number1} / {number2} = {result}")
25     else:
26         print(f"Désolé, je ne connais pas l'opération '{operation}'")
27
```

The IDE interface includes several panels:

- VARIABLES:** Shows local variables: `number1: 34.0`, `number1_string: '34'`, `number2: 23.0`, `number2_string: '23'`, and `should_continue: True`.
- WATCH:** Currently empty.
- CALL STACK:** Shows the current frame: `<module> s02e03.py 12:1`.
- BREAKPOINTS:** Shows two breakpoints: one at line 1 (checked) and one at line 12 (checked).
- TERMINAL:** Displays the program's output: `Je vais évaluer un calcul pour vous.`, `Tapez le premier nombre: 34`, and `Tapez le second nombre: 23`.

Red arrows point from the text labels to the corresponding parts of the IDE: from the first text to the variables panel, from the second text to the code editor, and from the third text to the breakpoints panel.

Mode

Style du code

Conventions seulement, mais utile pour s'y retrouver. [Suivez-les!](#)

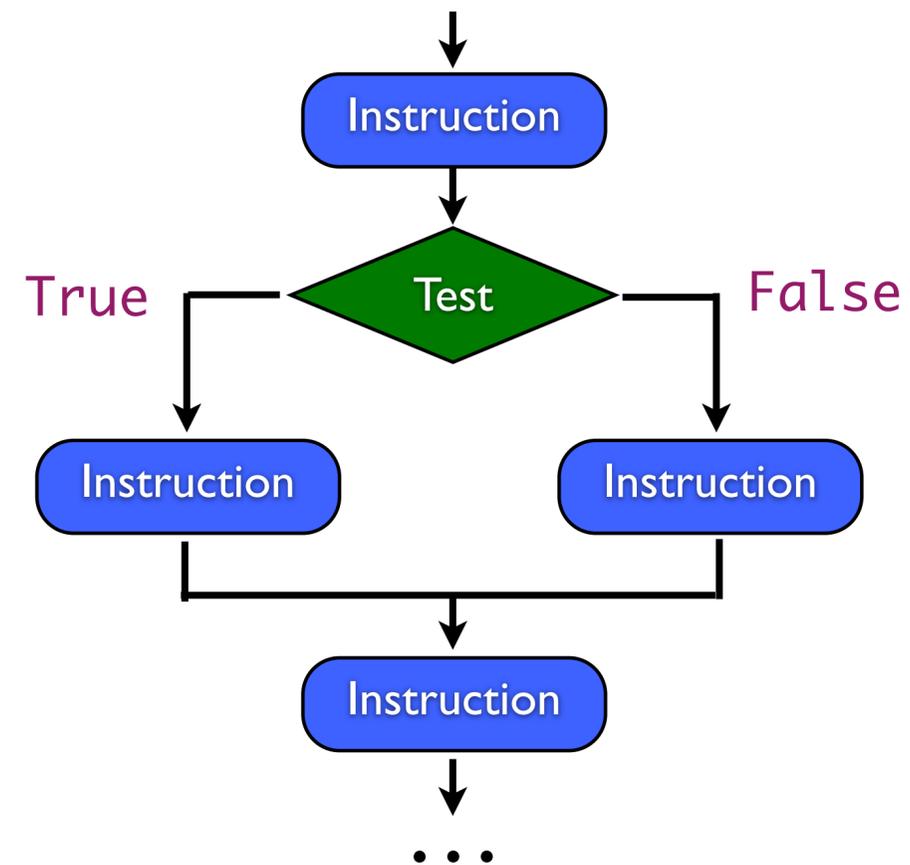
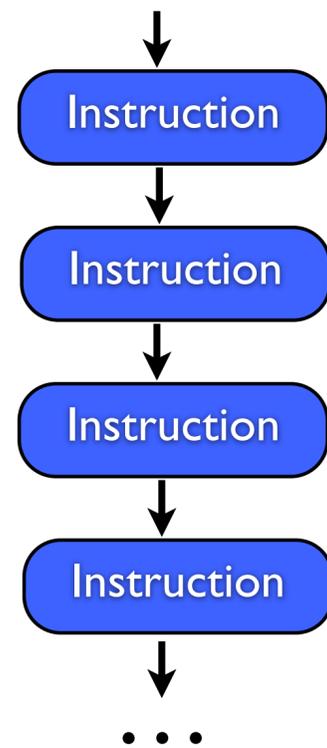
- **Nom des variables et méthodes**
 - Pas d'espaces! Pas de caractères spéciaux ou accentués
 - Pas de majuscules
 - Des underscores pour séparer les mots
 - Exemples: `age`, `my_name`, `first_name`
- **Indentation**
 - Nombre d'espaces ou tabs avant le début de la ligne
 - Change la signification du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

Cours de cette semaine

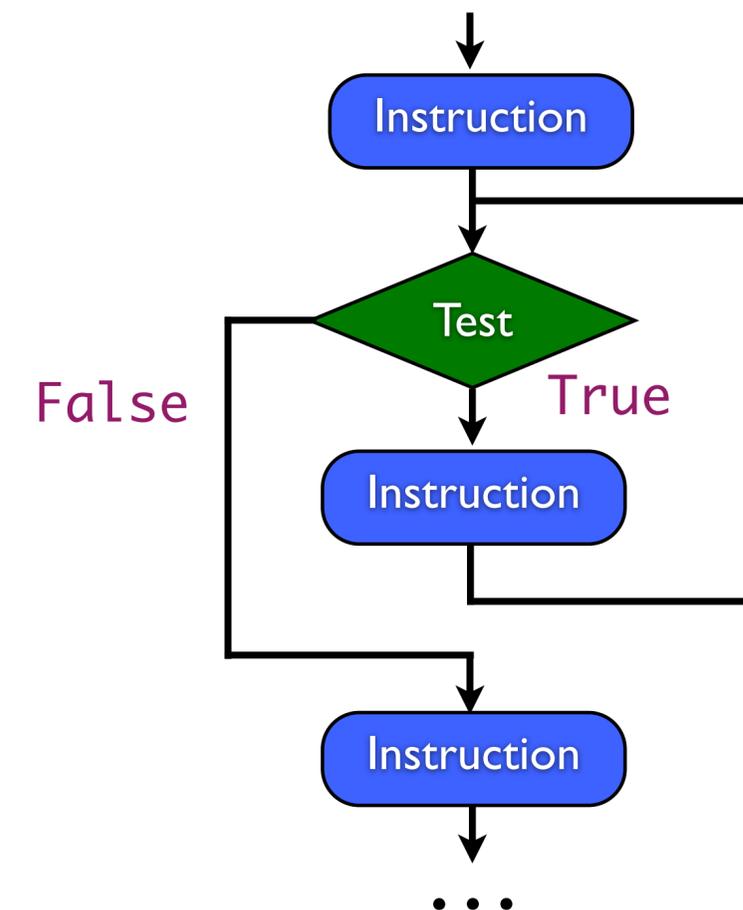
Conditions
Boucles

Motivation

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

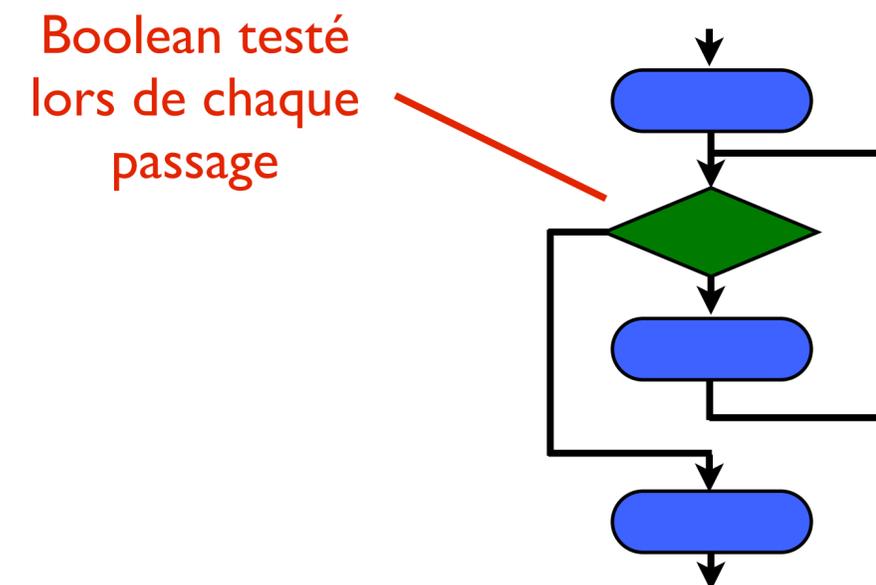
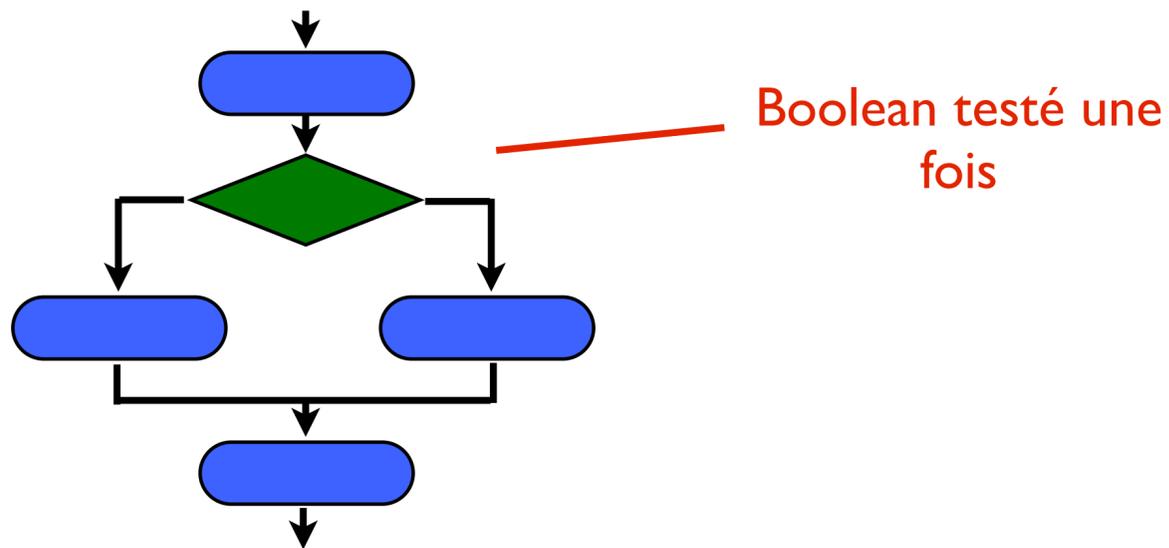


- **Boucles:**
«Affiche tous les multiples de 7 plus petits que 100.»



Conditions et boucles

- Des éléments de base de la programmation, 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

```
i = 7
while i < 100:
    print(i)
    i = i + 7
```

Initialisation

Test

Que faire à chaque itération

Mise à jour de la variable de boucle

Début
Changement
Fin

Pour chaque valeur (appelée *i*) dans ce *range*

```
for i in range(7, 100, 7):
    print(i)
```

Première valeur

Borne supérieure non incluse

Incrément

Que faire à chaque itération

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*