STRUCTURONS

À la fin de ce chapitre, je sais :

- ☞ lire et interpréter un code Python structuré
- ☞ indenter et enchaîner correctement les blocs Python
- ☞ coder une structure conditionnelle (et l'expression conditionnelle)
- ☞ coder une boucle avec un range sur l'intervalle d'entiers ouvert [0,n[
- ☞ coder une boucle tant que en précisant la condition de sortie
- ☞ coder une fonction (paramètres formels, effectifs, retour)

A Anatomie d'un programme Python

■ **Définition 1 — Instruction.** Une instruction dans un programme informatique est un mot qui explique à l'ordinateur l'action qu'il doit exécuter.

■ **Définition 2 — Programme informatique.** Un programme informatique est une suite d'instructions.

Un programme informatique se présente sous la forme d'un fichier texte organisé en parties et sous-parties, comme le montre le code 1.

Code 1 – Anatomie d'un programme Python (variables globales, fonctions, indentation, blocs, programme principal)

```
1  from random import randint
2
3  N = 42                # Global variables
4
5  def produit(a, b):    # BEGIN BLOC :
6      p = 0             # INDENTATION local variable
7      c = 0             # INDENTATION local variable
8      while c < a:      # INDENTATION instruction BEGIN BLOC :
9          p = p + b     # INDENTATION INDENTATION instruction
10         c = c + 1     # INDENTATION INDENTATION instruction
11     return p          # INDENTATION instruction
12
```

```
13
14
15 if __name__ == "__main__":     # MAIN PROGRAM
16     length = 10
17     results = []
18     while len(results) < length:
19         p = produit(randint(1, N), randint(1, N))
20         if p > N:
21             results.append(p)
22     print(results)
23     # --> [168, 280, 261, 740, 150, 507, 242, 288, 99, 84]
```

P Une des caractéristiques principales du langage Python est de donner un sens à l'indentation dans le code, c'est à dire que **les espaces en début de ligne ont une signification** : ils désignent un bloc d'instructions.

■ **Définition 3 — Bloc d'instructions Python.** Une suite d'instructions indentées au **même niveau** constituent un bloc d'instructions.

■ **Définition 4 — Programme principal.** Le programme principal est le point d'entrée de l'exécution d'un programme informatique : c'est par la que tout commence.

P En Python le programme principal s'appelle `__main__`. Il peut être introduit par les instructions `if __name__=="__main__":`. L'intérêt de cette instruction est de permettre au fichier de se comporter soit comme un programme à exécuter, soit comme un module que l'on peut alors importer dans un autre programme sans exécuter le programme principal.

B Programmation structurée

■ **Définition 5 — Programmation structurée.** Dans le cadre de la programmation structurée, un programme informatique peut-être développé à partir de trois structures :

1. les séquences d'instructions,
2. les structures alternatives,
3. les structures itératives.

Ce paradigme de programmation est une déclinaison du paradigme impératif. Il est proche du fonctionnement des processeurs actuels.

a Séquences d'instructions

Tous les langages de programmation dispose d'un moyen pour exprimer une séquence d'instruction. La plupart du temps on utilise des accolades (en Java, en C), parfois des délimi-

teurs de type begin ... end (Ocaml). En Python, le poitn virgule ; et l'indentation (cf. définition 3) permettent de constituer une séquence d'instruction.

b Structures conditionnelles

■ **Définition 6 — Instruction de test ou test.** Une instruction de test est une opération dont le résultat est un booléen.

■ **Définition 7 — Instruction conditionnelle.** L'exécution d'une instruction conditionnelle est soumise à la validité d'un test effectué en amont de l'instruction.

P En Python, il existe deux structures alternatives : le bloc `if then elif else` ou bien l'évaluation sous condition d'une expression `expr if then else`.

Bloc conditionnel en Python

Le code 2 recense les syntaxes possibles.

R Lorsqu'on utilise les structures conditionnelles, il est nécessaire d'être vigilant à ce que les conditions s'excluent mutuellement. Dans le cas contraire, c'est le premier test validé qui déclenche le branchement et l'exécution d'une instruction conditionnelle. On peut utiliser `if` sans `else` ou/et sans `elif`, mais il faut être logique dans l'élaboration des conditions pour que l'exécution aboutisse à un algorithme cohérent.

Dans l'exemple 2, il n'est pas souhaitable par exemple que les tests soient écrits ainsi `if age <= 50:` et `elif 50 <= age <= 60:`. Logiquement, cela signifierait que si `age` valait 50, on devrait afficher `"Still young"` ! **et** `"Has to work..."`. Or, seule la première chaîne de caractères sera affichée dans ce cas.

Code 2 – blocs conditionnels

```
1  age = 57
2  if age < 50:
3      print("Great !")
4
5  if age < 50:
6      print("Still young !")
7  else:
8      print("Not dead yet !")
9
10 if age < 50:
11     print("Still young !")
12 elif 50 <= age <= 67:
13     print("Has to work...")
14 else:
15     print("Get some rest !")
```

Expressions conditionnelles en Python

Cette syntaxe est à rapprocher du paradigme fonctionnel. Elle permet de conditionner l'évaluation d'une expression. Le code 3 décrit la syntaxe Python.

Code 3 – Expression conditionnelle

```
autruche = True
age = 45 if autruche else 46
print(age)  # 45
```

c Structures itératives

■ **Définition 8 — Structures itératives ou boucles.** Les structures itératives ou boucles permettent de répéter une séquence d'instructions un certain nombre de fois.

On distingue les boucles conditionnelles des boucles inconditionnelles.

Boucles inconditionnelles POUR

Les boucles inconditionnelles permettent de répéter une séquence d'instructions un nombre explicite de fois. Cela signifie que, lorsqu'on programme une boucle inconditionnelle, on connait ce nombre et on peut donc l'écrire dans le code.

Algorithme 1 Produit de deux nombres entiers avec boucle inconditionnelle

1 : $a \leftarrow 3$
2 : $b \leftarrow 4$
3 : $p \leftarrow 0$
4 : **pour** k de 0 à $b-1$ **répéter** ▷ b est un entier.
5 : $\quad p \leftarrow p+a$

Code 4 – Boucle inconditionnelle for

```
a = 3
b = 4
p = 0
for k in range(b):
    p += a
print(p)  # 12
```

P En Python, la fonction `range` permet d'exprimer la plage de variation de la variable de boucle. **Les paramètres de cette fonction sont des entiers** et il faut être vigilant car ceci est une source majeure de pertes de points à l'épreuve d'informatique commune. Le résultat est une séquence immuable.

La signature de cette fonction est `range(start, stop[, step])`. Si on omet de préciser start, alors on commence la séquence par zéro. Si on omet de préciser step, le pas par défaut vaut un. **La valeur stop est toujours exclue de la séquence.** Donc `range(10)` va renvoyer une séquence `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]` contenant **dix** éléments, comme le montre le code 5.

Le paragraphe précédent est important pour l'épreuve d'informatique!

Code 5 – Utilisation de range

```
print("Simple range")
for i in range(10):
    print(i, end="\t")      # 0 1 2 3 4 5 6 7 8 9

print("\nStart Stop range")
for i in range(3, 10):
    print(i, end="\t")      # 3 4 5 6 7 8 9

print("\nStart Stop Step range")
for i in range(1, 10, 2):
    print(i, end="\t")      # 1 3 5 7 9
```

Boucles conditionnelles TANT QUE

Les boucles conditionnelles permettent de conditionner la répétition d'une séquence d'instructions à une condition exprimée par un test. Cela signifie que, lorsqu'on programme une boucle conditionnelle, on ne sait pas nécessairement le nombre de répétitions à exécuter. Mais, on sait exprimer la condition d'arrêt des répétitions.

Algorithme 2 Produit de deux nombres entiers avec boucle inconditionnelle

1 : $a \leftarrow 3$ ▷ On veut calculer $a \times b$
2 : $b \leftarrow 4$
3 : $k \leftarrow 0$ ▷ Initialiser la variable de boucle
4 : $p \leftarrow 0$ ▷ p contiendra le résultat
5 : **tant que** $k < b$ **répéter**
6 : $p \leftarrow p + a$
7 : $k \leftarrow k + 1$ ▷ Incrémenter la variable de boucle

Code 6 – Boucle conditionnelle while

```
a = 3
b = 4
k = 0
p = 0
while k < b :
    p += a
    k += 1
print(p)  # 12
```

R D'une manière générale, on privilégiera l'utilisation des boucles `for`. On choisira une boucle `while` dans les cas où :

- l'on ne peut pas exprimer simplement le nombre de répétitions nécessaires,
- les conditions d'arrêt mettent en jeux des nombres flottants,
- on doit à la fois balayer tout un ensemble et respecter une certaine condition d'arrêt.

d Pourquoi compte-t-on à partir de 0 en informatique ? --→ HORS PROGRAMME

Cette question mérite qu'on s'y attarde un peu car, de nouveau, ceci est à l'origine de pertes de points significatives lors de l'épreuve d'informatique commune. Cette question est liée à une autre : pourquoi spécifie-t-on le paramètre `stop` dans la fonction `range` comme la borne supérieure plutôt que le maximum de la séquence ? Deux questions qui peuvent empêcher de dormir...

Dans une lettre restée célèbre [**dijkstra_why_1982**], Dijkstra [1] explique les raisons théoriques qui ont poussé les informaticiens à adopter cette convention, car il s'agit bien là d'une convention, on aurait pu choisir de procéder différemment.

La convention choisie pour délimiter les bornes d'une séquence de nombres entiers est la suivante : $\min \leq i < \sup$, le minimum étant inclus et la borne supérieure exclue. Par exemple, la séquence $[0,1,2,3,4,5,6,7,8,9,10,11,12]$ est exprimée $0 \leq i < 13$.

Tout d'abord, Dijkstra observe qu'une séquence de nombres entiers naturels possède toujours un plus petit élément par lequel on commence ou finit. Donc l'exclure n'est pas judicieux. Ceci justifie le choix du symbole $\leq$. Ensuite, il observe qu'on pourrait très bien commencer à indexer les séquences par l'élément numéro 1. Mais si on adopte cette convention, alors la plage des indices s'exprime $1 \leq i < N+1$, ce qui n'est pas très élégant et ne donne pas d'emblée la longueur de la séquence. Par contre, si on commence à zéro, alors cette plage des indices s'écrit $0 \leq i < N$ ce qui présente à la fois l'avantage :

- d'indiquer directement la longueur de la séquence (N),
- de ne pas à avoir à ajouter en permanence +1 pour préciser les plages d'indices.

Une autre raison, plus électronicienne, milite pour commencer à zéro. Un tableau en mémoire commence à une case numérotée. Quand on va chercher dans la mémoire le premier élément du tableau, on va donc récupérer cette case. Si on souhaite récupérer le second élément, on incrémente le numéro de la case mémoire de un. Le premier élément se trouve donc à l'indice 0 et le second à l'indice 1.

C'est pourquoi, comme Dijkstra, il faut considérer zéro comme le nombre entier le plus naturel pour commencer à dénombrer ! D'ailleurs, le mot *zéro* vient de l'italien *zefiro* et qui provient lui-même de larabe *sifr* qui signifie à l'origine le vide, le néant. Quoi de plus naturel que le vide pour commencer ?

1. prononcer DailleKeStra

C Fonctions Python

Dans un programme, Il est inutile voire dangereux de répéter du code déjà écrit : cela nuit à la fiabilité [2], à la lisibilité [3] et à la généricité. C'est pour éviter cela que le paradigme procédural a été inventé.

■ **Définition 9 — Paradigme procédural.** Le paradigme procédural incite à regrouper dans un même code les fonctionnalités importantes d'un programme dans un but d'intelligibilité et de réutilisabilité du code écrit. Les fonctionnalités sont regroupés dans des routines ou des procédures.

P En Python, les routines sont appelées fonctions. On les appellera donc ainsi par la suite.

■ **Définition 10 — Fonctions Python.** Une fonction Python est caractérisée par :

- son nom,
- ses paramètres formels,
- une valeur de retour (optionnelle).

■ **Définition 11 — Appel d'une fonction.** Appeler une fonction, c'est demander son exécution. Cela s'effectue en respectant le prototype de la fonction et en utilisant l'opérateur `()` qui demande l'exécution de la fonction à l'interpréteur.

■ **Définition 12 — Prototype d'une fonction.** Le prototype d'une fonction sert de mode d'emploi au développeur. Dans la plupart des langages, le prototype d'une fonction est constitué du nom de la fonction, des paramètres formels qu'elle accepte et de la valeur renvoyée.

■ **Définition 13 — Paramètres formels.** Ce sont les paramètres qui sont utilisés pour écrire la fonction. Ils sont situés dans le prototype. Ils sont formellement exigés par la fonction pour opérer.

■ **Définition 14 — Paramètres effectifs.** Ce sont les paramètres qui apparaissent lors de l'appel d'un fonction, c'est à dire lorsqu'on utilise la fonction. Il sont effectivement transmis à la fonction.

■ **Exemple 1 — .** Le code 7 donne un exemple très simple de fonction. Sur cet exemple, on peut noter :

- le prototype de la fonction : `square(a)`,

2. car on peut se tromper en recopiant
3. car cela allonge le code inutilement

- le paramètre formel `a`,
- la valeur renvoyée `x`,
- le paramètre effectif `3`.

Code 7 – Fonction Python à un paramètre renvoyant un entier

```
1 def square(a):      # prototype, a is a formal parameter
2     x = a * a       # function body
3     return x        # returned value
4
5 nine = square(3)    # 3 is an effective parameter (argument)
6 print(f"Squaring three gives {nine}") # Squaring three gives 9
```

Vocabulary 1 — Parameters - arguments ⇜⇝ En anglais, les paramètres formels sont désignés par le terme *parameters*, les paramètres effectifs par *arguments*. Certains emploient ces mots en français également.

Le code 8 montre un exemple de procédure, c'est à dire une fonction qui ne renvoie rien. Elle ne fait, en apparence seulement, aucun calcul qu'on puisse sauvegarder. Par contre, elle interagit avec la console, c'est à dire avec l'écran de la machine via le système d'exploitation, pour faire afficher des chaînes de caractères [4].

Code 8 – Procédure Python

```
1 def say_hello(name, n):    # prototype
2     for i in range(n):     # function body
3         print(f"Hello {name} !", end="\t")
4         # no returned value
5
6 say_hello("Olivier", 2) # Hello Olivier ! Hello Olivier !
```

Le code 9 montre un exemple de fonction avec paramètre optionnel. Si le paramètre `capitalized` n'est pas fourni lors de l'appel de la fonction alors celui-ci se voit attribuer la valeur `True`. S'il existe un paramètre effectif lors de l'appel, c'est celui-ci qui est utilisé.

Ce code montre également qu'avec une structure conditionnelle, il peut exister plusieurs chemins d'exécution avec une valeur retour différente.

Code 9 – Fonction avec paramètre optionnel et plusieurs chemin d'exécution avec valeur retour

```
1 def say_my_name(name: str, capitalized=True):
2   # signature with optional parameter
3     if capitalized:
4         return name.capitalize()    # returned value
5     else:
6         return name.upper()         # returned value
7
8 my_name = say_my_name("olivier")    # calling function
```

4. ce que l'on appellera un effet au second semestre.

```
print(my_name)                          # Olivier
my_name = say_my_name("olivier", False)     # calling function
print(my_name)                              # OLIVIER
```

D Conventions de nommage

■ **Définition 15 — Conventions.** Règles de conduite adoptées à l'intérieur d'un groupe social. Exemple : en mathématiques, l'inconnue c'est x.

D'une manière générale, en informatique contemporaine, on préfère les conventions aux configurations [5], car cela fait gagner un temps précieux. Respecter des conventions dans l'écriture des programmes permet leur traitement automatique par d'autres outils pour générer d'autres programmes (tests, vérification, documentation, mise en ligne, interfaçage). Cela permet également aux développeurs de comprendre plus rapidement un code.

L'intelligibilité est certainement la plus grande qualité qu'on puisse exiger d'un code. Aujourd'hui, dans la plupart des entreprises, des normes de codage sont appliquées afin de rendre le travail collaboratif plus efficace. Pour le langage Python, un ensemble de recommandations a été produit (cf. Python Enhancement Proposals - PEP 8). Donc, si vous ne savez pas trop comment faire, il est toujours possible de s'y référer. Les IDE actuels implémentent ces recommandations et peuvent formater ou proposer des changements conformes aux Python Enhancement Proposals.

M **Méthode 1 — Choisir un nom de variable ou de fonction** *Mal nommer les choses c'est ajouter au malheur de ce monde* [a]. Alors tâchons de bien les nommer.

À faire :

- choisir des noms de variables et de fonctions en minuscules.
- si plusieurs mots sont nécessaires, mettre un _ entre les mots,
- préférer un verbe pour les fonctions,
- appeler un chat un chat.

À ne pas faire :

- utiliser les lettres l,O ou I pour un nom de variables. Elles sont confondues avec 1 ou 0.
- utiliser autre chose que les caractères ASCII (par exemple des lettres accentuées, des lettred grecques ou des kanjis),
- appeler une variable d'après un mot clef du langage Python (lambda, list, dict, set, global, try, True...)

a. citation apparemment de Brice Parrain dans une réflexion sur l'étranger d'Albert Camus. À vérifier cependant.

5. c'est à dire des arrangements spécifiques.

■ **Exemple 2 — Noms courants de variables.** Voici des noms possibles et courants pour :

- les types `int` : `i,j,k,m,n,p,q,a,b`
- les types `float` : `x,y,z,u,v,w`
- les accumulateurs : `acc, s, somme, prod, produit, product`
- les pas [a] : `dt, dx, dy, step, pas,`
- les chaînes de caractères : `c, ch, s,`
- les listes : `L, results, values,`
- les dictionnaires : `d, t, ht,`
- les constantes en majuscules : `MAX_SIZE`.

a. c'est à dire la distance entre chaque élément d'un vecteur temporel par exemple

M **Méthode 2 — Faire un commentaire** Un commentaire peut être utile dans une copie pour détailler un point du code ou expliquer un choix d'implémentation que vous avez fait. Mais d'une manière générale, il n'est pas forcément nécessaire, pourvu que votre code soit intelligible. Le choix des noms des variables, le respect de l'indentation et des conventions sont les clefs d'un code intelligible.

Que peut-il arriver de pire à quelqu'un qui raconte une blague? Devoir l'expliquer. Il en est de même du commentaire. Le commentaire peut être utile à l'intelligibilité mais il peut lui nuire également. Par ailleurs, dans le temps, un commentaire peut faire référence à une ligne modifiée ou une variable dont le nom a changé. Dans ce cas là, le commentaire nuit à la compréhension.

On s'attachera donc à :

- n'inscrire que des commentaires utiles et brefs,
- s'assurer qu'ils sont cohérents avec le code,
- à ne pas paraphraser le code.

On préfèrera écrire un code directement lisible. Sur le site de ce cours, vous trouverez des exemples de copies de concours que je vous aide à analyser. Bien écrire et bien nommer permet de gagner des points.