Programmation avec Python

Module 02 : Les fondamentaux du langage

eni

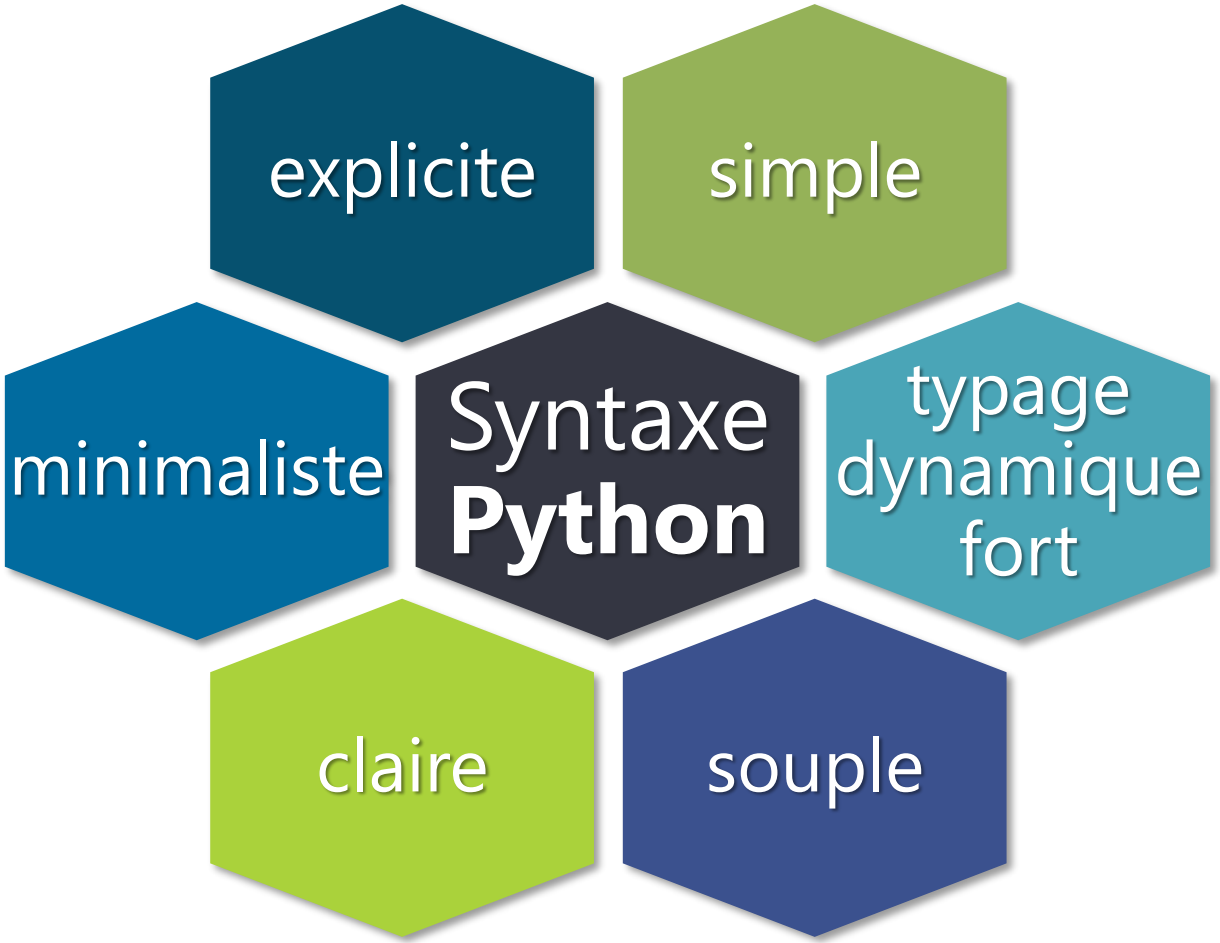
OBJECTIFS

- Connaître la structure d'un programme
- Savoir manipuler variables et chaînes de caractère
- S'approprier les opérateurs
- Connaître les structures de contrôle

SOMMAIRE

« Zen de Python »

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!



- Fin d'instruction : touche « Entrée » ⏎
- Bloc d'instructions :
  - Ouverture d'un bloc : « Deux Points »
  - Intégration par indentation : « 4 Espace » ou « Tabulation » ⇆
- Commentaire :
  - `# ceci est un commentaire de ligne`
  - `" Ceci est une chaine de caractère et commente sur une ligne "`
  - `""" Ceci est un commentaire de type paragraphe sur plusieurs lignes. """`

Veiller à ne pas mélanger les espaces et les tabulations pour l'indentation du code.

LES NOMS DE VARIABLE

- Le nom d'une variable doit être composé uniquement de :
  - Lettres (minuscule, majuscule)
  - Chiffres (0…9)
  - Symbole « espace souligné » (_)
- Ne doit pas commencer par un chiffre.
- Ne doit pas être un mot-clé Python :

```
# Liste des mots-clés
import keyword
print(keyword.kwlist)
```

['False', 'None', 'True', 'and', 'as', 'assert', **'async'**, **'await'**, 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

- Sensible à la casse (minuscule / Majuscule)
- Convention : **`lower_snake_case`** (pour la lisibilité)

Même s'ils sont tolérés, éviter les caractères accentués (encodage).

DÉCLARER UNE VARIABLE

```
nom_de_la_variable = valeur

autre_variable # NameError: name *** is not defined


nom_de_la_variable:type = valeur
# accepté, mais purement informel.
```

L'affectation d'une valeur à une variable est obligatoire.

```
a = 12 et b = 34.78
Addition : 12 + 34.78
Résulat =  46.78
Traceback (most recent call last):
  File "<string>", line 14, in <module>
ERROR!
NameError: name 'result' is not defined
```

```python
# Déclaration de variables :
a, b = 12, 34.78
# Affichage des variables 'a' et 'b' :
print("a =", a, "et b =", b)

def add(val1, val2):
    print("Addition :", val1, "+", val2) # Affichage des arguments
    result = val1 + val2 # variable locale
    return result # Retourne le résultat du calcul
 # Fin du bloc 'additionner'

somme = add(a, b) # Appel de la fonction avec val1=a et val2=b
print("Résultat = " , somme) # Affichage de la variable 'somme'


print("Variable result =" , result)
print("val1=", val1, "et val2=", val2)
```

LES TYPES PRINCIPAUX

Contenu		Code (Type)	Ex. Valeur	Précision
Booléen		**`bool`** (boolean)	• ***`True`*** • ***`False`***	• Deux états seulement
Numérique	Entier	**`int`** (integer)	-123 456789	• Négatif/Positif (±) • Aucune limite ∞ *(/ architecture machine)*
	Décimal	**`float`** (floating point number)	-123.456…789 789.0123…456	• Négatif/Positif (±) • Séparateur décimal (•) • Aucune limite ∞ *(/ architecture machine)*
Chaîne de Caractères		**`str`** (string)	'Trop "Fun" !' '''Paragraphe''' "J'adore" """Paragraphe"""	Délimitations : • apostrophe (**'**) • guillemet (**"**) • 3x apostrophes • 3x guillemets

EXEMPLE DE TYPES DE BASE

```
# Booléen :
IS_TRUE = True
IS_FALSE = False

# Numérique Entier :
LIMIT_AGGLO = 50
limit_dep = 80

# Numérique avec Décimal :
ext_temperature = 37.8
PI_APPROX_6 = 3.141592

# Chaîne de Caractères :
code = 'H4cK3r'
SAINT_EXUPERY = "On ne voit bien qu'avec le coeur"
```

LES CONSTANTES

- Python ne possède pas de concept de constante
  - Rappel : « Tout est modifiable dans Python »
- Convention : *UPPER_SNAKE_CASE* ou *SCREAMING_SNAKE_CASE*
  - Exemple :
    - `SCHOOL_NAME = "ENI – École Informatique"`
    - `AGGLOMERATION_SPEED_LIMIT = 50`
    - `LIMIT_TEMPERATURE = 38.2`

Il s'agit que d'une convention, cela ne provoquera pas d'erreur de compilation.

Opérateur	Symbole(s)	Exemple	Résultat
Addition	+	1 + 2	3
Soustraction	-	5.4 - 3.2	2.2
Multiplication	*	4 * 6	24
Division	/	9 / 2	4.5
Modulo	%	9 % 2	1
Division Entière	//	9 // 2	4
Puissance	**	3 ** 2	9

- Séquences d'instructions
- Conditionnelles
- Répétitives

LES OPÉRATEURS D'AFFECTATION

Opérateur	Signe(s)	Utilisation	Equivalent
Affectation	=	a = b	
Addition	+=	a += b	a = a + b
Soustraction	-=	a -= b	a = a - b
Multiplication	*=	a *= b	a = a * b
Division	/=	a /= b	a = a / b
Modulo	%=	a %= b	a = a % b
Division Entière	//=	a //= b	a = a // b
Puissance	**=	a **= b	a = a ** b

- Opérateur de concaténation (+)

```
# Concaténation (explicite) :
print("Hello " + "World !")

print("Typage fort = " + 321) # Pas de conversion implicite str/int|float

print("Numérique = " + str(654)) # Conversion explicite

# Concaténation implicite via print :
print("Hello", "the", "World", \
    "from", "Python", 3, "!")
```

```
Hello World !
Traceback (most recent call last):
  File "...\Concatenation.py", line 4, in <module>
    print("Typage fort = " + 321) ...
Numérique = 654
Hello the World from Python 3 !
```

- Fonction de conversion de variable suivant leurs types :

```
an_int = int(input("Saisir un nb. entier : ")) # str -> int
a_float = float(input("Saisir un nb. Décimal (.) : ")) # str -> float

an_int = int(un_float) # float -> int (perte de la partie décimale)
a_float = float(un_int) # int -> float (non nécessaire)

a_str = "Résultat = " + str(an_int + a_float) # int ou float -> str
```

- En cas d'erreur de conversion (Traceback) :
  - **ValueError: invalid literal for int() with base 10: 'x.yy'**
  - **ValueError: could not convert string to float: '???'**
  - **TypeError: can only concatenate str (not "float") to str**

```
# Saisie depuis la console d'une valeur (type texte) :
first_name = input("Veuillez saisir votre prénom : ")

# Affichage de la valeur :
print("Bonjour", first_name)

a_number = input("Choisissez un nombre entre 1 et 100 : ")

print("Vous avez choisi le nombre", a_number)
```

Veuillez saisir votre prénom : Kenny

Bonjour Kenny

Choisissez un nombre entre 1 et 100 : 28

Vous avez choisi le nombre 28

`input()` renvoie systématiquement une chaine de caractères.

DÉMONSTRATION

Lire et écrire dans la console

- Opérateurs de comparaison :

Opérateur	Signe(s)	Exemple	Résultat
Strictement Supérieur	`>`	`12 > 34`	`False`
Supérieur ou égal	`>=`	`12.56 >= 34.7`	`False`
Strictement Inférieur	`<`	`12 < 34`	`True`
Inférieur ou égale	`<=`	`12.56 <= 34.7`	`True`
Égal	`==`	`2.0 == 5.4`	`False`
Différent	`!=`	`2.0 != 5.4`	`True`

- Opérateurs de logique :

Opérateur	Signe(s)	Exemple	Résultat
Et (booléen)	`and`	`True and True`	`True`
Ou (booléen)	`or`	`True or False`	`True`
Non (booléen)	`not`	`not False`	`True`
Et logique *(binaire)*	`&`	`True & False`	`False`
Ou logique *(binaire)*	`|`	`True | True`	`True`

- Syntaxe : **`if` *`condition`*`:` …**

```
# Vérification de l'âge
txt_age = input("Quel est votre âge ? ")
age = int(txt_age) # Conversion du texte en nb. entier

AGE_ADULT = 18 # Limite entre une personne majeure/mineure

is_adult = False # Valeur par défaut

if age >= AGE_ADULT:
    is_adult = True

print("Vous êtes majeur :", is_adult)
```

```
Quel est votre âge ? 38
Vous êtes majeur : True
...
Quel est votre âge ? 15
Vous êtes majeur : False
```

- Syntaxe : **if** *condition*: … **else**: …

```
# Vérification de l'âge
txt_age = input("Quel est votre âge ? ")
age = int(txt_age) # Conversion du texte en nb. entier

AGE_ADULT = 18 # Limite entre une personne majeure/mineure

if age >= AGE_ ADULT:
    adult_or_child = "majeur"
else:
    adult_or_child = "mineur"

print("Vous êtes", adult_or_child, "!")
```

Quel est votre âge ? 32
Vous êtes donc majeur !

...

Quel est votre âge ? 12
Vous êtes donc mineur !

- Syntaxe : **if** *condition1*: … **elif** *condition2*: … **else** : …

```
# Vérification de l'âge
age = int(input("Quel est votre âge ? ")) # Conversion en nb. entier

AGE_ADULT = 18 # Limite entre une personne majeure/mineure
AGE_SENIOR = 70 # Âge d'un jeune sénior

if age >= AGE_SENIOR:
    category_age = "senior"
elif age >= AGE_ADULT:
    category_age = "majeur"
else:
    category_age = "mineur"

print("Vous êtes donc un", category_age, "!")
```

```
Quel est votre âge ? 78
Vous êtes donc un senior !
...
Quel est votre âge ? 14
Vous êtes donc un mineur !
...
Quel est votre âge ? 35
Vous êtes donc un majeur !
```

- Il n'existe pas de mécanismes tel que le *« switch case »*,
- Il est recommandé d'utiliser la forme complète du *« if…elif…else »*,
- Des alternatives sont possibles à partir de *dictionnaire {…}*.

DÉMONSTRATION

Les conditionnelles

TRAVAUX PRATIQUES

Les conditionnelles

- Syntaxe : **`while`** *condition*:
- Bloc itératif conditionné :

```
print("Table de Multiplication")
nb = int(input("Indiquer un nombre : "))
# Affichage de la table de multiplication

# Code avec une boucle while
i = 0
while i < 10:
    i += 1
    print(i, "x", nb, "=", i * nb)
# Fin du bloc itératif
```

Table de Multiplication
Indiquer un nombre : 6
1 x 6 = 6
2 x 6 = 12
3 x 6 = 18
4 x 6 = 24
5 x 6 = 30
...
9 x 6 = 54
10 x 6 = 60
_____Fin du programme

- Syntaxe générique :

```
for nb_cycle in range(cycle_max):
```

- Bloc itératif séquencé :

```
print("Table de Multiplication")
nb = int(input("Indiquer un nombre : "))
# Affichage de la table de multiplication

# Code équivalent avec une boucle for…in…range
for i in range(10):
    m = i + 1
    print(m, "x", nb, "=", m * nb)
# Fin du bloc itératif
```

```
Table de Multiplication
Indiquer un nombre :  6
1 x 6 = 6
2 x 6 = 12
3 x 6 = 18
4 x 6 = 24
5 x 6 = 30
...
9 x 6 = 54
10 x 6 = 60
_____Fin du programme
```

- Possibilité d'utiliser **`continue`** pour passer à l'itération suivante, et **`break`** pour sortir d'une boucle.

- Syntaxe générique :

```
for nb_cycle in range(min, max):
```

- Bloc itératif séquencé :

```
print("Table de Multiplication")
nb = int(input("Indiquer un nombre : "))
# Affichage de la table de multiplication

# Code équivalent avec une boucle for…in…range
for i in range(1, 11):
    print(i, "x", nb, "=", i * nb)
# Fin du bloc itératif
```

Table de Multiplication
Indiquer un nombre : 6
1 x 6 = 6
2 x 6 = 12
3 x 6 = 18
4 x 6 = 24
5 x 6 = 30
…
9 x 6 = 54
10 x 6 = 60
_____Fin du programme

DÉMONSTRATION

Les boucles

TRAVAUX PRATIQUES

Les boucles