Types simples

1 - Nombres : int et float

1.1 Encoder une information

Encoder une information veut dire qu'on transforme notre information en **séquence de 0 et de 1** en respectant une **technique d'encodage** précise. Si on connaît la technique d'encodage, on peut **décoder**.

Information -> ENCODAGE -> 010101011100...

010101011100... -> DECODAGE -> Information

1.2 Integers

Les **entiers** sont encodés sous forme d'**integer**. Nommé `int` en Python. La fonction native `type()` indique le type.

```
>>> type(5)
<class 'int'>
```

```
>>> type(5.0)
<class 'float'>
```

Les **signatures associées** sont indiquées ci-dessous.

▷ **Addition** avec `+` : `int + int -> int`

▷ **Soustraction** avec `-` : `int - int -> int`

▷ **Multiplication** avec `*` : `int * int -> int`

▷ **Division** avec `/` : `int / int -> float`

▷ **Quotient** avec `//` : `int // int -> int`

▷ **Reste** avec `%` : `int % int -> int`

▷ **Puissance** avec `**` : `int ** int -> int`

```
>>> 23 / 5
4.6        # Attention au float sur /
```

1.3 Floats

Les **nombres à virgule** sont encodés par le type `float`. X symbolisera un opérateur parmi `+ - * / // % **`.

— `float X float -> float`

— `float X int -> float`

— `int X float -> float`

Dès qu'il y a un float en entrée, la signature indique un float en sortie.

1.4 Le problème des flottants

Integer Les entiers sont **exactement enregistrés**.

`7 -> ENCODAGE -> 00000111 -> DECODAGE -> 7`

Float Certains floats sont enregistrés **approximativement**. Lorsqu'on décode, on obtient donc parfois **presque** la valeur.

`0.1 -> ENCODAGE -> 00..010101.. -> DECODAGE -> 0.10000000000000001`

A SAVOIR IMPERATIVEMENT : l'encodage des nombres réels en float provoque parfois un enregistrement approximatif de la valeur de départ.

Conséquence : Si on peut faire les calculs en se passant des flottants, on privilégie cette solution.

Le plus perturbant : parfois c'est exact, parfois c'est approximatif. Nous verrons pourquoi dans DONNEES.

```
>>> 4.8 - 4.2
0.5999999999999996

>>> 3 * 0.1
0.30000000000000004
```

```
>>> 0.75 - 0.25
0.5

>>> 3 * 0.25
0.75
```

Comme vous le voyez, cela provoque **parfois** des erreurs de calculs. Avec plusieurs milliards d'approximations par seconde, on peut obtenir rapidement des erreurs énormes.

2 - Texte : string (str)

2.1 Définition

Le **string** est un conteneur formant une collection ordonnée de caractères.

Indice	0	1	2	3	4	5	6
Caractère	'B'	'o'	'n'	'j'	'o'	'u'	'r'

Première case 0 : avec 7 cases, les indices vont de 0 à 6.

Indice se traduit par **index** en anglais.

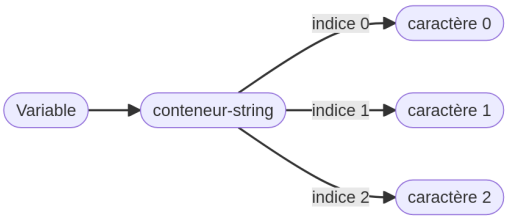

FIGURE 4.1 – Principe du string

String est un mot anglais qui signifie chaîne : un texte est un ensemble de caractères qui se suivent. Exemple avec le mot **Bonjour** :

2.2 Déclaration avec Python

Les strings sont gérés par le type **str**.

Déclaration

Un string Python est délimité par deux indicateurs signalant le début et la fin du string. Cet indicateur peut être :

— un guillemet simple : **'Bonjour'** (touche du 4)
— trois guillemets simples : **"'Bonjour"'**
— un guillemet double : **"Bonjour"** (touche du 3)
— trois guillemets doubles : **"""Bonjour"""**

Jamais de mélange pour un même string : si on commence avec un double, on finit avec un double.

```
>>> "Hello World !'
  File "", line 1
    "Hello World !'
                  ^
SyntaxError: EOL while scanning string literal
```

Python répond toujours avec des guillemets simples lorsqu'il le peut.

Déclaration d'un string multi-ligne

On peut déclarer un string intégrant des passages à la ligne (celles intégrant des appuis sur ENTREE) en utilisant 3 guillemets :

```
1  s = """Voici une liste :
2  - premier truc
3  - deuxième truc
4  - troisième truc
5
6  En ici, c'est fini"""
7
8  print(s)
```

Voici l'affichage obtenu dans la console :

```
Voici une liste :
- premier truc
- deuxième truc
- troisième truc

En ici, c'est fini
```

Déclaration d'un string vide

Deux solutions :

```
>>> s = ""
>>> s
''
```

```
>>> s = str()
>>> s
''
```

2.3 Opérateurs Python

▷ **Concaténation** avec **+** : **str + str -> str**

La concaténation correspond à une sorte d'addition, mais la mécanique utilisée est celle d'un juxtaposition. D'où l'utilisation d'un autre terme.

```
>>> "bon" + "jour"
'bonjour'
```

```
>>> '5' + '5'
'55'
```

```
>>> 'Bonj' + 5
TypeError: can only concatenate str to str
```

▷ **Répétitition** avec ***** :

— **int * str -> str**
— **str * int -> str**

La répétition correspond à une sorte de copier-coller d'un string.

```
>>> "Bon" * 2
'BonBon'
```

```
>>> 4 * "Pom ! "
'Pom ! Pom ! Pom ! Pom ! '
```

```
>>> '#' * 20
'####################'
```

▷ **Les autres ?** : les autres opérateurs ne sont pas implémentés sur les strings. Leur utilisation avec un string provoque donc une erreur.

2.4 Déterminer sa longueur

La longueur d'un string correspond au nombre de "cases-caractères" du string. On utilise la fonction native **len()**.

```
Indice     0123456
>>> mot = "Bonjour"
>>> len(mot)
7
```

Il y a 7 caractères, on sait alors qu'on peut demander des indices allant de 0 à ...6.

3 - Propriété : bool

Propriété affirmation qui peut être vraie ou fausse.

Booléens

Lorsqu'on utilise certains opérateurs booléens, l'interpréteur évalue votre expression en répondant **True** si c'est vrai, **False** sinon.

Opérateurs renvoyant un booléen

▷ **Strictement supérieur >** :

Python gére cet opérateur avec des nombres, des strings...

```
>>> 40 > 30
True

>>> "crocodile" > "tortue"
False
```

On utlise l'ordre lexicographique (du dictionnaire si tout en minuscule).

▷ **Egalité** avec **==**

```
>>> 40 == 3 * 10
False

>>> 40 == 4 * 10
True

>>> 0.3 == 3 * 0.1
False
```

▷ **Différence** avec **!=**

```
>>> 40 != 3 * 10
True

>>> 40 != 4 * 10
False
```

▷ **Autres opérateurs de comparaison**

— Supérieur ou égal **>=**

— Inférieur ou égal **<=**

▷ **Appartenance avec in**

Le mot-clé **in** permet de savoir si un élément **a** est présent dans un élément **b** : **a in b**

```
>>> "soja" in composition
False
```

4 - Transtyper : passer d'un type à un autre

```
# 4.4.1 VIDE

>>> i = int()
>>> i = 0

>>> f = float()
>>> f = 0.0

>>> s = str()
>>> s = ''

>>> b = bool()
>>> b = False
```

```
# 4.4.3 FLOAT

>>> f = float("5")
>>> f
5.0

>>> f = float("5.2")
>>> f
5.2

>>> f = float("")
ValueError
```

```
# 4.4.2 INTEGER

>>> i = int(5)
>>> i
5

>>> i = int(5.2)
>>> i
5

>>> i = int("5")
>>> i
5

>>> i = int("5.2")
ValueError

>>> i = int("")
ValueError

>>> i = int(False)
>>> i
0

>>> i = int(True)
>>> i
1
```

```
# 4.4.4 STRING

>>> s = str(5)
>>> s
'5'

>>> s = str(5.2)
>>> s
'5.2'

>>> s = str(False)
>>> s
'False'

>>> s = str(True)
>>> s
'True'
```

```
# 4.4.5 BOOLEEN

>>> b = bool(0)
>>> b
False

>>> b = bool(5)
>>> b
True

>>> b = bool(5.2)
>>> b
True
```

```
# 4.4.6 CLAVIER

>>> n = int(input())

>>> n = float(input())
```

A SAVOIR PAR COEUR Un contenu VIDE/NUL est évalué à **False** si on veut un booléen. Sinon, c'est **True**.

www.infoforall.fr

CC BY NC SA