Types simples

4.1

Nombres : int et float

4.1.1 Encoder une information

Encoder une information veut dire qu'on transforme notre information réelle en **séquence de 0 et de 1** en respectant une **technique d'encodage** précise

Information -> ENCODAGE -> 010101011100...

Si on connaît la technique utilisée pour créer cette séquence, on pourra alors la **décoder**.

010101011100... -> DECODAGE -> Information

4.1.2 Integers

Les **entiers** sont encodés sous forme d'**integer**. En Python, ce type est nommé `int`. On peut trouver le type d'une donnée en utilisant la fonction native `type()`.

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

On retrouve les opérateurs vus dans Python 2. Voici les **signatures associées**.

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

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

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

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

`float ?` C'est le type utilisé pour les nombres à virgule. Notez bien qu'en Python, la virgule (notation française) est représentée par un point (notation anglo-saxonne).

```
>>> 23 / 5
4.6
```

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

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

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

4.1.3 Floats

Les **nombres à virgule** sont encodés sous un type de données nommé `float` en Python.

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

▷ **Addition** avec `+` :

- `float + float -> float`
- `float + int -> float`
- `int + float -> float`

▷ **Soustraction** avec `-` :

- `float - float -> float`
- `float - int -> float`
- `int - float -> float`

▷ **Multiplication** avec `*` :

- `float * float -> float`
- `float * int -> float`
- `int * float -> float`

▷ **Division** avec `/` :

- `float / float -> float`
- `float / int -> float`
- `int / float -> float`

▷ **Puissance** avec `**` :

- `float ** float -> float`
- `float ** int -> float`
- `int ** float -> float`

Un peu perturbant : les opérateurs de division euclidienne qui renvoient un float dès lors qu'on trouve un float parmi les entrées.

▷ **Quotient** avec `//` :

- `float // float -> float`
- `float // int -> float`
- `int // float -> float`

▷ **Reste** avec `%` :

- `float % float -> float`
- `float % int -> float`
- `int % float -> float`

Dès qu'il y a un flottant en entrée, la réponse sera sous forme d'un flottant.

4.1.4 Le problème des flottants

Integer Tous les nombres sont toujours **exactement enregistrés** : on parvient à relire exactement ce qu'on a enregistré.

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

Float Certains nombres stockés sous forme d'un float sont enregistrés **approximativement**. Lorsqu'on décode, on obtient donc parfois presque la valeur qu'on a voulu enregistrée. Presque.

`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 la partie 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 : en toute rigueur 4.8 - 4.2 donne exactement 0.6 en mathématique et 3 * 0.1 donne exactement 0.3 en mathématique.

Multipliez ces approximations par plusieurs milliards d'opérations bancaires par seconde dans le monde, et vous imaginerez sans peine qu'il y aurait des erreurs énormes à bout de quelques temps.

4.2

Texte : string (str)

4.2.1 Définition

Le type permettant de gérer les textes se nomme souvent **string**. Il s'agit d'un conteneur formant une collection ordonnée d'éléments qui sont tous des caractères.

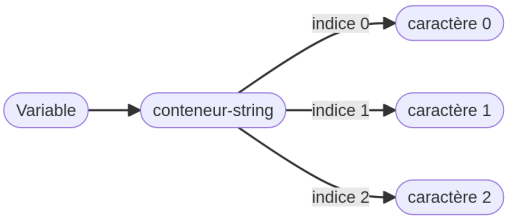

FIGURE 4.1 – Principe du string

String est un mot anglais qui signifie "chaîne" en français. Or, un texte est une chaîne de caractères : un ensemble de caractères qui se suivent. Exemple avec le mot `Bonjour` :

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

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

Première case : la première case est la case d'indice 0, pas celle d'indice 1. Les indices d'un string de 7 caractères vont donc de 0 à 6.

4.2.2 Déclaration avec Python

En Python, les strings sont gérés par le type nommé `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
''
```

4.2.3 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.

4.2.4 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.

4.3 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...

— **`int|float > int|float -> bool`**

— **`str > str -> bool`**

Le signe | veut dire OU : entier ou flottant.

```
>>> 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`**

Si la réponse est **`True`**, c'est que **`a`** apparaît dans **`b`**. Sinon, la réponse sera **`False`**. Tester si quelqu'un d'allergique au soja peut manger un produit :

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

Convertir : passer d'un type à un autre

4.4.1 Variable "vide"

Création d'un contenu "vide" mais typée.

Création d'un contenu vide

```
>>> i = int() # integer "vide"
>>> i
0

>>> f = float() # float "vide"
>>> f
0.0

>>> s = str() # string "vide"
>>> s
''

>>> b = bool() # booléen "vide"
>>> b
False
```

Déclaration directe

```
>>> i = 0     # integer "vide"
>>> f = 0.0   # float "vide"
>>> s = ""    # string "vide"
>>> b = False # booléen "vide"
```

4.4.2 Créer un integer

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

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

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

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

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

>>> i = int("")
ValueError

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

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

4.4.3 Créer un float

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

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

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

>>> f = float("")
ValueError

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

>>> f = float(True)
>>> f
1.0
```

4.4.4 Créer un string

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

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

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

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

>>> s = str("bonjour")
>>> s
'bonjour'

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

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

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

4.4.5 Créer un booléen

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

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

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

>>> b = bool(0.0)
>>> b
False

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

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

>>> b = bool("bonjour")
>>> b
True

>>> b = bool("")
>>> b
False

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

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

A SAVOIR PAR COEUR Un contenu VIDE/NUL est évalué à `False` si on veut un booléen. Sinon, le résultat est `False`.