NSI - Première

Python - 2 - Types simples

qkzk

Les types

En Python chacun des objets qu'on manipule et donc chacune des variables qu'on écrit a un type.

On accède au type d'une expression avec la fonction `type` qui s'emploie ainsi :

```
>>> type(123)
<class: 'int'>
```

Le type d'une variable permet de définir ce qu'il est possible de faire avec cette variable.

Par exemple, on peut ajouter deux entiers, on peut mesurer la longueur d'une chaîne de caractère, mais on ne peut pas mesurer la longueur d'un entier ou ajouter un entier et une chaîne de caractère.

Les nombres

Python distingue plusieurs ensembles de nombres :

- les entiers (sous partie finie des entiers relatifs),
- les flottants (sous partie finie des nombres réels),

Les entiers : le type `int`

On peut réaliser toutes les opérations courantes sur les entiers mais attention à la division :

```
>>> 4 / 3
1.3333333333333
>>> 4 // 3
1
```

- La division flottante est obtenue par l'opération `/`
- La division entière par l'opération `//`

Lorsque `a` et `b` sont du type entier, les opérations : `+, -, *, //, %, **` sont définies et le résultat sera un entier.

La division de `a / b` donnera toujours un flottant.

Les nombres à virgule flottante : le type `float`

C'est une approximation des nombres réels. Nous étudierons leur construction et leur représentation en mémoire plus tard dans l'année.

Les opérations courantes sur les flottants sont elles-aussi définies et ce type l'emporte généralement sur les entiers.

Plus précisément, cela signifie que `1 + 2.3` sera du type flottant.

Exercice 1

1. Sans importer la librairie `math`, comment calculer la racine carré d'un nombre ? Rappel : $x^{0.5} = \sqrt{x}$ pour tout $x \geq 0$.
2. On considère un triangle ABC rectangle en A, calculer la longueur BC sachant que $AB = 4$ et $AC = 5$.

Le module `math`

Un module est un ensemble de fichiers python qu'on peut importer lorsqu'on en a besoin.

On importe un module avec `import nom` (sans extension `.py`)

Par exemple :

```
>>> import math             # importe le module des fonctions mathématiques
>>> dir(math)               # affiche son contenu
['__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb',
'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp',
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd',
'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm',
'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter',
'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt',
'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

On utilise ensuite le module en l'appelant par son nom :

```
>>> math.pi
3.141592653589793
```

On peut accéder à l'aide d'une fonction avec la fonction `help`

```
>>> help(math.cos)
Help on built-in function cos in module math:

cos(x, /)
    Return the cosine of x (measured in radians).
```

On quitte l'aide avec la touche 'Q'.

L'aide donne une information utile, pour mesurer les angles, il faut utiliser les radians.

Exercice 2

1. Utilisez la fonction `math.radians` pour mesurer le cosinus de 45°.
2. Que fait la fonction `math.sqrt` ?
3. Reprendre la question 2 de l'exercice 1 en utilisant le module `math` et sans utiliser l'opérateur `**`.
4. Documentez-vous sur la fonction `hypot` et testez la. Proposez une troisième version de cet exercice à l'aide de cette fonction.

Les booléens, le type `bool` : `True` et `False`

- Il existe deux booléens : `True` et `False`,
- Le résultat d'une opération de comparaison est toujours un booléen.

```
>>> type(True)
<class 'bool'>
>>> 1 + 1 == 2
True
>>> type(2 + 2 == 5)
<class 'bool'>
>>>
```

Les opérations courantes sur les booléens sont `not` qui renvoie le contraire, `and`, le *et* logique et `or`, le *ou* inclusif.

```
>>> not True
False
>>> (True and not False) or (False and True)
True
```

Exercice 3

1. Vérifier à la main le résultat précédent.
2. Pour accéder à l'attraction "Autopia" du parc Disneyland, un enfant doit être agé de 6 ans au moins, mesurer 81 cm au moins s'il est accompagné ou mesurer 1m30 au moins s'il n'est pas accompagné.

   On considère les variables :

   - `taille` (`int` en cm),
   - `age` (`int` en année),
   - `est_accompagne` (`bool`)

   Proposer une expression booléenne qui permette de savoir si un enfant peut entrer dans le parc.

   Tester **tous** les cas.

On considère un triangle ABC de longueurs respectives x, y et z.

2. Proposez (sur une feuille d'abord !) une expression booléenne sur les variables `x`, `y`, `z` (`float`, supérieurs ou égaux à 0) qui permette de vérifier l'inégalité triangulaire : *la longueur d'un côté est inférieure à la somme des longueurs des deux autres côtés.*
3. Traduire cette expression booléenne en une variable booléenne `est_un_triangle` et testez la.

Les chaînes caractère, le type `str`

Définition

Une chaîne de caractère est une succession de caractères, symboles afficheables à l'écran.

```
>>> "Salut"
'Salut'
>>> 'la forme ?'
'la forme ?'
```

On définit généralement une chaîne de caractère en la saisissant entre des guillemets ou des apostrophes. Il faut utiliser le même symbole pour ouvrir et fermer la chaîne :

```
>>> "bonjour'
  File "<stdin>", line 1
    "bonjour'
             ^
SyntaxError: EOL while scanning string literal
```

Disposer de deux notations `"` et `'` permet d'insérer des guillemets ou des apostrophes :

```
>>> "j'ai 120 ans"
"j'ai 120 ans"
```

Remarquez que cette fois Python utilise des `"` dans la valeur !

Exerice 4

1. Créer une variable contenant la phrase :

Il m'a dit qu'il m'aimait !

2. Recommencez avec la phrase :

   Je lui ai dit : "pas moi !"

Opérations courantes

Il existe de nombreuses opérations sur ce type, les plus courantes sont :

- mesurer la longueur avec `len`
- concaténer (mettre bout à bout) avec `str + str`
- répéter avec `str * int`

```
>>> len("super")
5
>>> "salut" + "les" + "amis"
'salutlesamis'
>>> "salut" * 5
'salutsalutsalutsalutsalut'
```

Les autres combinaisons de `+` et `*` sont impossibles :

```
>>> 'salut' * 'nsi'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can't multiply sequence by non-int of type 'str'
>>> 'salut' + 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Ajoutons un autre opérateur, l'appartenance : `in`.

```
>>> nom = "Ducobu"
>>> "u" in nom
True
>>> "z" in nom
False
>>> "cobu" in nom
True
```

`a in b` est vrai si `a` est une sous-chaîne de `b`.

Exercice 5

1. Testez la concaténation avec les phrases de l'exercice précédent. Que se passe-t-il lors de l'affichage ?
2. *La disparition* de Georges Perec est un roman célèbre. En voici un extrait :

   > Puis, à la fin, nous saisirons pourquoi tout fut bâti à partir d'un carcan si dur, d'un canon si tyrannisant. Tout naquit d'un souhait fou, d'un souhait nul : assouvir jusqu'au bout la fascination du cri vain, sortir du parcours rassurant du mot trop subit, trop confiant, trop commun, n'offrir au signifiant qu'un goulot, qu'un boyau, qu'un chas, si aminci, si fin, si aigu qu'on y voit aussitôt sa justification.

   Vérifiez à l'aide de Python qu'une lettre très courante ne figure pas dans ce passage.

Formatage

Python permet de *formater* les chaînes de caractères, c'est à dire d'y insérer une variable en lui donnant une représentation précise.

Il existe de nombreuses technique mais nous allons nous contenter des “f-strings” :

```
>>> prenom = "Marcel"
>>> age = 12
>>> f"Je suis {prenom} et j'ai {age} ans"
"Je suis Marcel et j'ai 12 ans"
```

Python a remplacé pour nous les variables `prenom` et `age` par leur valeur.

Si on oublie le `f`, Python ne fait rien :

```
>>> "Je suis {prenom} et j'ai {age} ans"
"Je suis {prenom} et j'ai {age} ans"
```

Exercice 5

En utilisant les variables `nom`, `prenom`, `age` et `phrase` (une f-string) écrire une phrase similaire à :

> Je m’appelle Raoul Ducobu et j’ai 65 ans.

Accéder à un caractère particulier

On peut accéder à un caractère en utilisant les `[ ]`.

Entre les crochets on indique l’*indice* du caractère souhaité :

```
>>> prenom = "Célia"
>>> prenom[0]       # le 1er caractère est d'indice 0
'C'
>>> prenom[1]       # le second caractère et d'indice 1
'é'
>>> prenom[23]      # plus grand que la taille, va provoquer une erreur
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Pour accéder de manière systématique au dernier caractère d’une chaîne (ou plus tard d’une autre collection d’objets indexés), on peut utiliser `len` :

```
>>> prenom = "Raoul"
>>> dernier_caractere = prenom[len(prenom) - 1]
>>> dernier_caractere
'l'
```

Python permet aussi de se répérer à partir de la fin, ce qui raccourcit la notation précédente en :

```
>>> prenom = "Raoul"
>>> dernier_caractere = prenom[-1]
>>> dernier_caractere
'l'
```

Exercice 6

1. Saisissez vos noms et prénoms dans des variables et créez une variable avec vos initiales. `Paul Quesnoy -> PQ`
2. Recommencez pour former la notation abbrégée suivante : `P. Quesnoy`

Mutabilité

Un objet est *mutable* s'il contient des éléments qu'on peut modifier.

Les chaînes caractères *ne sont pas mutables*.

C'est une notion fondamentale que vous devez vous efforcer de retenir. Elle n'a guère d'importance pour l'instant mais sera source de nombreuses erreurs ultérieures si vous ne la comprenez pas.

Cela signifie qu'on ne peut pas changer le contenu d'une chaîne de caractère.

```
>>> animal = "Lyon"          # zut une faute d'orthographe !
>>> animal[1] = "i"          # on essaie de rectifier
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> animal = "Lion"          # pas le choix, je dois écraser la précédente variable
```

Nous rencontrerons bientôt deux autres types similaires aux chaînes de caractères : les `list` et les `tuple`. Les `list` sont mutables et les `tuple` ne le sont pas.

Caractères spéciaux

Certains caractères ne sont pas afficheables directement à l'écran, d'autres ont des significations particulières.

Par exemple un `\n` dans une chaîne de caractère signifie qu'il sera remplacé par un retour à la ligne lors d'un affiche age.

```
>>> phrase = "j'ai faim\nje mange"
>>> phrase
"j'ai faim\nje mange"
>>> print(phrase)
j'ai faim
je mange
```

notez la différence :

```
>>> phrase
```

Python affiche *la valeur* de la variable : `"j'ai faim\nje mange"`

```
>>> print(phrase)
```

Python n'affiche pas la valeur de `print(...)` mais c'est `print` elle-même qui affiche quelque chose.

`print` est donc à réserver aux pure opérations d'affichages :

- lorsqu'on programme et qu'on souhaite connaître la valeur d'une espression rapidement,
- lorsqu'on crée une interface *texte* dans laquelle on affiche des informations à l'utilisateur.

Méthodes et méthodes sur les chaînes de caractères.

On a rencontré déjà deux fonctions : `len` et `print`. Il existe une autre variété de fonctions, appelées *méthodes* et qui s'appliquent à un objet lui même.

Considérons un exemple : la méthode `upper`.

```
>>> "bonjour".upper()
'BONJOUR'
>>> help(str.upper)
Help on method_descriptor:

upper(self, /)
    Return a copy of the string converted to uppercase.
```

Comme on peut le voir :

- elle s'utilise avec *la notation pointée* : `chaine.upper()`
- elle ne prend pas de paramètre (en fait si, `self`, l'objet *lui même* mais c'est pour l'an prochain),
- elle renvoie une *copie* de la chaîne dans laquelle toutes les lettres sont mises en majuscule.

Exercice 7

1. Vérifiez qu'après l'exécution de la méthode `upper`, l'objet n'a pas changé.
2. Documentez vous sur les méthodes `count`, `lower` et `replace` des chaînes de caractères. Que font-elles ?
3. Testez les. Combien de "a" sont présents dans l'extrait de la disparition ?