Slides modified from **https://github.com/UGE-IGM/courspython**

Chapitre 1 : Valeurs, types et variables

Dans ce chapitre vous allez :

1. découvrir des types Python de base, apprendre à les reconnaitre et les utiliser ;
2. connaitre quelles opérations sont supportées pour chaque types en jeu ;
3. apprendre à convertir d'un type vers un autre ;
4. découvrir la notion de variable et le mécanisme d'affectation ;
5. apprendre à faire saisir une donnée à l'utilisateur et utiliser sa réponse.

I Types de valeurs

Toutes les valeurs en Python possèdent un **type**. Le type d'une valeur définit les **opérations** qu'il est possible de lui appliquer.
Les types de base sont :

- les nombres entiers (`int`)
- les nombres décimaux (`float`)
- les booléens (`bool`)
- les chaines de caractères (`str`)
- un type de valeur indéfinie (`NoneType`)

Nombres entiers (`int`)

In [1]:

```
983
```

Out[1]:

```
983
```

In [3]:

```
1234
```

Out[3]:

```
1234
```

In [4]:

```
-4  # un entier négatif
```

Out[4]:

```
-4
```

In [5]:

```
2 ** 10  # un très très grand entier !!!
```

Out[5]:

1024

Nombres décimaux (`float`)

In [ ]:

```
3.14
```

In [6]:

```
-1.5
```

Out[6]:

```
-1.5
```

In [7]:

```
.1
```

Out[7]:

```
0.1
```

In [8]:

```
12.
```

Out[8]:

```
12.0
```

In [9]:

```
4.56e3  # notation scientifique
```

Out[9]:

4560.0

Booléens (`bool`)

Ce type permet de représenter les deux valeurs de vérité « vrai » et « faux ». On en reparlera au prochain chapitre.

In [1]:

```
True   # vrai
```

Out[1]:

```
True
```

In [ ]:

```
False  # faux
```

Attention : Les majuscules/minuscules sont importantes :

In [2]:

```
true # provoque une exception (une erreur)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_21928/1293955843.py in <module>
----> 1 true # provoque une exception (une erreur)

NameError: name 'true' is not defined
```

Chaînes de caractères (`str`)

Une chaine de caractères est une succession de symboles (lettres, chiffres, ou autres) entre des guillemets.
En Python, il y a plusieurs moyen d'en construire pour répondre à la nécessité d'avoir à l'intérieur d'elle même des guillemets.

La construction des chaines de caractères

In [6]:

```
'asdbonjour'  # guillemets simples
```

Out[6]:

```
'asdbonjour'
```

In [ ]:

```
"hello !"  # guillemets doubles
```

In [ ]:

```
"மனிதப் பிறவியினர் சகலரும் சுதந்திரமாகவே பிறக்கின்றனர்."  # caractères non-latins
```

"""Pour des chaînes plus longues (par exemple un paragraphe entier) on peut utiliser des guillemets triples."""

Il faut faire un peu attention pour écrire une chaîne de caractères contenant des apostrophes ou des guillemets :

In [1]:

```
print("hello, my namne is whatever")
```

```
hello, my namne is whatever
```

In [ ]:

```
'Mon nom est "Personne".'
```

Affichage de chaîne

In [ ]:

```
print("Bonjour à toi, Ô lecteur attentif !")
```

Caractères spéciaux : \n, \t, ', ", \

In [ ]:

```
"sauts\nde\nligne"  # \n provoquera un passage à la ligne si on affichait la chaine de caractère
```

In [10]:

```
print("sauts\nde\nligne")
```

```
sauts
de
ligne
```

In [ ]:

```
print("Du\ttexte\nsur\t2 colonnes")  # touche ⇥
```

In [ ]:

```
print("D'autres symboles spéciaux : \' \" \\")
```

Valeur indéfinie (NoneType)

In [ ]:

```
None  # ça a l'air de ne servir à rien mais en fait c'est bien pratique
```

Vérifier vos connaissances

A ce stade, vous devez être capable :

- de citer les cinq types Python de base que nous avons vu ;
- de reconnaitre le type d'une expression simple ;
- de donner les trois manières de créer une chaine de caractères ;
- d'écrire des chaines de caractères contenant elle même des guillemets ;
- de citer les principaux caractères spéciaux qu'une chaine peut contenir et d'expliquer leur effet ;
- savoir comment afficher une chaines de caractères.

Exercice 1 : Validité et correction de chaines de caractères

Les chaines de caractères suivantes sont-elles bien construites ? Si non, les corriger.
Remarque : *Il se peut que vous ayez à faire des choix et qu'il y ait plusieurs corrections possibles.*

In [ ]:

```
""Bonjour""
```

In [ ]:

```
"\"Ceci est une citation\""
```

In [ ]:

```
"Suis je une chaine ?'
```

In [ ]:

```
"Je contiens une \tabulatio\n"
```

In [ ]:

```
'"Ceci est encore une citation"'
```

II Opérations

Le type d'un objet détermine les **opérations** qu'on peut lui appliquer.

Opérations sur les nombres

Addition (`a + b`), soustraction (`a - b`), multiplication (`a * b`), puissance (`a ** b`)

- sur deux `int` et produisant un `int`
- ou sur deux `float` et produisant un `float`
- ou sur un `int` et un `float` et produisant un `float`

In [ ]:

```
4 + 5
```

In [5]:

```
4 - 5.
```

Out[5]:

```
-1.0
```

In [ ]:

```
4. * 5.
```

In [ ]:

```
4 ** 0.5  # 4 puissance 1/2 c'est la racine carrée de 4 !
```

Division "réelle" (`a / b`) : produit toujours un `float`

In [ ]:

```
1/3
```

In [ ]:

```
4 / 2  # ne donne pas un entier !!
```

Division euclidienne :

- **quotient** (`a // b`)
- **reste**, ou **modulo** (`a % b`)
- si `a` et `b` de type `int`, produisent un `int`, sinon un `float`

In [3]:

```
7 // 2
```

Out[3]:

```
3
```

In [ ]:

```
7 % 2
```

In [ ]:

```
divmod(7, 2)
```

In [ ]:

```
4 // 2  # cette fois c'est un entier...
```

In [ ]:

```
4 % 2  # le reste est nul car 4 est pair (divisible par 2)
```

In [ ]:

```
4.0 // 1.75  # donne un float !
```

In [ ]:

```
4.0 % 1.75
```

Les opérations suivent les règles de priorité usuelles :

In [ ]:

```
4 + 2 * 1.5
```

On peut aussi utiliser des parenthèses :

In [ ]:

```
(4 + 2) * 1.5
```

Opérations sur les chaînes de caractères

Concaténation : `s + t`

In [ ]:

```
'Universite' + 'Sorbonne' + 'Paris' + "Nord"
```

In [ ]:

```
'Universite' + ' Sorbonne' + ' Paris' + " Nord"
```

Répétition : `s * a`

In [ ]:

```
'Hip ' * 3 + 'Hourra !'
```

In [ ]:

```
('Hip ' * 3 + 'Hourra ! ') * 2
```

Beaucoup d'autres opérations (sur les chaînes, les nombres...) : *on verra ça plus tard*

Conversions / transformations de type

On a parfois besoin de convertir une valeur d'un type à l'autre :

- Une valeur en chaîne de caractères avec la fonction `str`

In [ ]:

```
"J'ai " + 10 + ' ans.'
```

In [ ]:

```
"J'ai " + str(10) + ' ans.'
```

- Un `float`, et *parfois* un `str` en `int`

In [ ]:

```
int(3.5)  # float vers int
```

In [ ]:

```
int('14')  # str vers int
```

In [ ]:

```
int('3.5')  # impossible : deux conversions (str -> float -> int)
```

In [ ]:

```
int('deux')  # impossible : ne représente pas un nombre
```

- Un `int`, et *parfois* un `str` en `float`

In [ ]:

```
float(3)  # int vers float
```

In [ ]:

```
float('14.2')  # str vers float
```

In [ ]:

```
float('3,5')  # impossible : virgule au lieu de point
```

In [ ]:

```
float('bonjour')  # impossible : ne représente pas un nombre
```

Accès au typage d'une expression

On peut accéder au type d'une expression grâce à la fonction prédéfinie `type`.

In [ ]:

```
type("salut")
```

In [ ]:

```
type(4 / 2)
```

In [ ]:

```
type(2 * 4.8)
```

Vérifier vos connaissances

A ce stade, vous devez être capable de :

- décrire les opérations disponibles en Python sur les nombres et de connaitre le type du résultat ;
- décrire les opérations disponibles en Python sur les chaines de caractères ;
- savoir convertir un type numérique en chaine de caractère, et inversement ;
- savoir comment accéder au type d'une valeur Python.

Exercice 2 : valeur et type d'une opération

Pour chacune des instructions suivantes :

1. donner le type et le résultat de l'expression donnée ;
2. vérifier le résultat.

On pourra utiliser la fonction `type` si nécessaire pour vérifier le type du résultat.

In [ ]:

```
2 * 5
```

In [ ]:

```
2 + 1.5
```

In [ ]:

```
2.0 * 4
```

In [ ]:

```
'2.0' * 4
```

In [ ]:

```
'2.0' * 4.0
```

In [ ]:

```
4 / 2
```

In [ ]:

```
4.0 / 2
```

In [ ]:

```
5 / 2
```

In [ ]:

```
5 % 2
```

In [ ]:

```
5 // 2
```

In [ ]:

```
int(4.0) / 2
```

In [ ]:

```
str(4) / 2
```

In [ ]:

```
'toto' + str(4)
```

In [ ]:

```
float(4) * 2
```

In [ ]:

```
int(str(4) * 2)
```

In [ ]:

```
'toto' + 'titi'
```

In [ ]:

```
int('toto') + 'titi'
```

In [ ]:

```
int(2.0) * 4
```

In [ ]:

```
'toto' * str(4)
```

In [ ]:

```
int('1.25')
```

III Variables et affectations

Une **variable** est un *nom* servant à désigner une valeur

- Une variable est remplacée par sa valeur dans les calculs
- Seules les opérations du type de la valeur sont permises

L'**affectation** est le fait de lier une *valeur* à une *variable*

- Syntaxe : `nom = expression`

Attention : Ce n'est pas *du tout* le = des mathématiques, il faut le lire comme "prend la valeur"

In [12]:

```
x = 3
y ='USPN'
z = x + 2
print(x)
print(y)
print(z)
```

```
3
USPN
5
UPSN
```

- On peut *réaffecter* une variable (même une valeur d'un type différent)

In [ ]:

```
print(x)
x = 'USPN'
print(x)
```

- On ne peut utiliser une variable que si elle a été préalablement définie !

In [ ]:

```
print(foo)
```

Étapes de l'affectation

AFFECTATION SIMPLE : `x = 40 + 2`

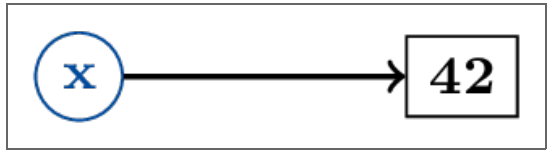
In [ ]:

```
x = 40 + 2
print(x)
```

1. Calcul du *membre droit* (ici `42`) et stockage dans la mémoire

2. Création du nom `x` (sauf si déjà créé)

3. Création du lien entre *variable* et *valeur*

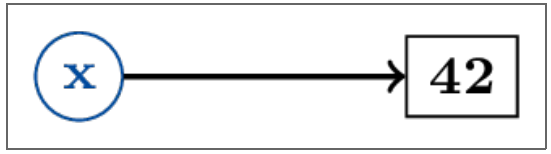


Deuxième exemple : `y = x`

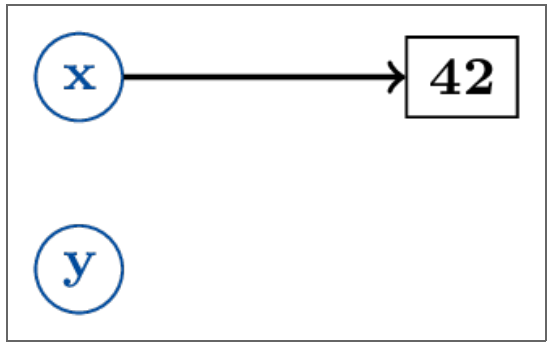
In [ ]:

```
y = x
print(x, y)
```

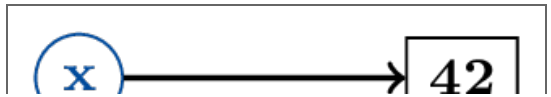
1. Calcul du *membre droit* après remplacement de chaque variable par la valeur associée (ici `x` remplacé par `42`)

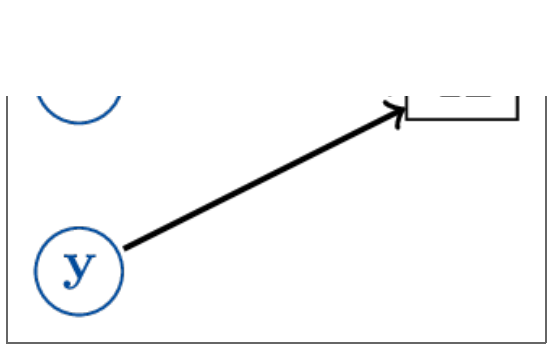


2. Création du nom `y` (sauf si déjà créé)



3. Création du lien entre `y` et `42`



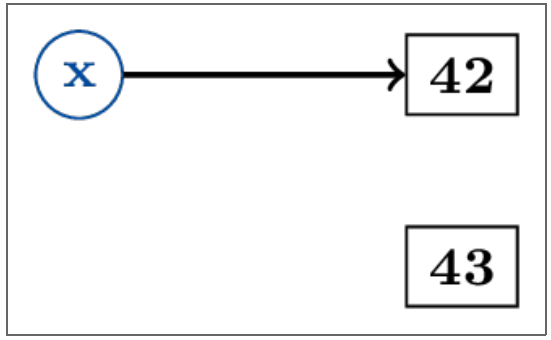

y

Troisième exemple : x = x + 1

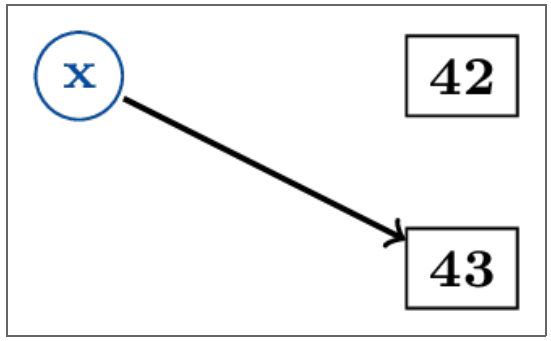
In [ ]:

```
x = x + 1
print(x)
```

1. Calcul du *membre droit* après remplacement de chaque variable par la valeur associée (ici `x` remplacé par `42`, résultat : `43`)



2. Nom `x` déjà existant, création du lien entre `x` et 43



Remarque : L'instruction `x = x + 1` peut aussi s'écrire `x += 1`. De même, `x *= 2` est une version rapide de `x = x * 2`.

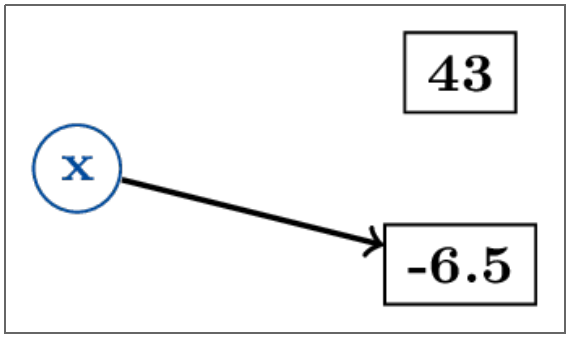
In [ ]:

```
x += 1
print(x)
```

Dernier exemple : `x = -6.5`

In [ ]:

```
x = -6.5
print(x)
```



Quelques points de détail :

- On peut réaffecter une variable avec une valeur de type différent (comme dans le dernier exemple)
- En cas de *réaffectation* d'une variable, le lien précédent est perdu.
- Si aucun autre lien n'existe vers la valeur précédente, celle-ci est "détruite" par le système de ramasse-miette (*garbage collector*)

Vérifier vos connaissances

A ce stade, vous devez être capable :

- d'expliquer ce qu'est une variable à quelqu'un ;
- d'expliquer ce qu'est l'opération d'affectation d'une variable ;
- dessiner l'état de la mémoire à l'issue d'une suite d'intructions simples.

Exercice 3 : Etat de la mémoire après une suite d'instruction

Dessiner l'état de la mémoire à l'issue des instructions suivantes :

```
x = 2
y = 3
x += y
y *= 2
```

Que vaut alors `x` et `y` ?

Nommage des variables

Règles de nommage des variables :

- Commencent par une *lettre*, suivie de *lettres et de chiffres*
- Le caractère *underscore* `'_'` est considéré comme une lettre
- Éviter les caractères spéciaux (accents, cédille, etc.)
- Les *mots réservés* (ou mots-clés) de Python sont interdits
- Il y a aussi des **conventions** (*vues plus tard*)

Exemples : `_ex2` `Ex2mpl1`
Contre-exemple : `2014mlv`

Mots-clés et autres mots réservés

Les mots suivants sont **réservés** pour le langage :

```
and     as       assert     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
    True    False    None
```

Au passage :

- Tout ce qui se trouve après un caractère `#` est un **commentaire**
- Mettre des commentaires dans le code est un bon moyen de pouvoir le reprendre facilement

Vérifier vos connaissances

A ce stade, vous devez être capable :

- de choisir un nom de variable correct et signifiant.

Exercice 4 : nommage de variables

Indiquer parmi les mots suivants ceux qui ne sont pas des noms valides pour une variable :

```
bonjour             Hi!                 au revoir           oui
Ciao                NON                 byeBye7             6hello6
abc                 def                 6hello6             _upem_
good_morning        __repr__            good-afternoon      f()
```

IV Lecture et affichage

Fonction de saisie : `x = input("Veuillez rentrer ...")`

- L'utilisateur tape une ligne au clavier
- La ligne est stockée sous forme de chaîne de caractères (`str`)
- Cette valeur peut ensuite être affectée à une variable (ici `x`)
- On peut aussi omettre le message d'invite pour l'utilisateur, mais cela devient moins convivial

Fonction d'affichage : `print(x)`

- Affiche dans le terminal la chaîne de caractères associée à `x`
- On peut afficher plusieurs valeurs d'un coup en les séparant par des virgules : `print(x, y, z, ...)`
- Appelle automatiquement la fonction `str` sur chacun de ses arguments
- S'il y a plusieurs arguments, insère automatiquement des espaces
- Passe automatiquement à la ligne

In [ ]:

```
prenom = input('votre prenom : ')
```

In [ ]:

```
prenom
```

In [ ]:

```
print("Bonjour", prenom, "!")
```

Remarque : *Il existe de nombreuses possibilités pour l'affichage de texte, consulter la* ***documentation officielle*** *pour plus de détails.*

Vérifier vos connaissances

A ce stade, vous devez être capable :

- d'afficher n'importe qu'elle chaine de caractère
- stocker dans une variable une donnée entrée par l'utilisateur, en la convertissant correctement

Exercice 5 : lecture d'entiers

Commenter ligne par ligne le programme suivant et indiquer ce qui est affiché si l'utilisateur saisit 4 :

In [13]:

```
r = input("Donnez moi un entier : ")
print(r + r)
n = int(r)
print(n + n)
```

```
Donnez moi un entier : 342
342342
684
```

Exercice 6 : affichage mixte

Corriger le programme suivant :

In [ ]:

```
nombre = 3
print("j'ai" + nombre + "pommes")
```