Université de Gabés
Faculté des Sciences de Gabés

LGLSI 1

Semestre 2

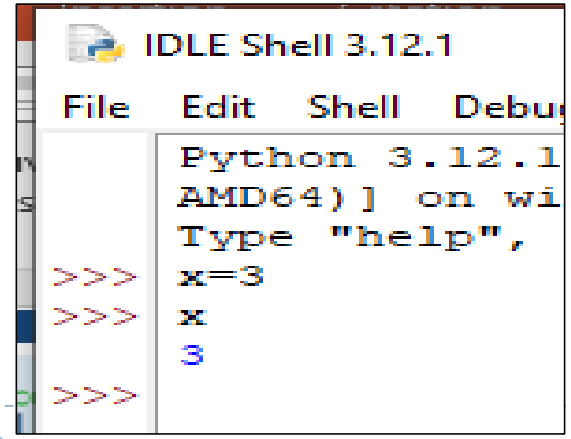
Chapitre 2 : Les variables

Dhikra KCHAOU
dhikrafsegs@gmail.com

2023-2024

Les variables dans le langage Python

- Une variable est caractérisée par :
  - Un identificateur (nom)
  - Une valeur
  - Un type
  - Une adresse
- En Python, **la déclaration** d'une variable et **son initialisation** se font en même temps.



Python a « deviné » que la variable est de type entier. On dit que Python est un langage doté d'un **typage dynamique**

Variables

- **Règles utilisées lors de la définition d'une variable**:
  - Le nom des variables peut être constitué de lettres minuscules (a à z), lettres majuscule (A à Z), nombres (0 à 9) et le caractère souligné (_)
  - Les variables « test», « Test » ou « TEST » sont différentes → **Python est sensible à la casse.**
  - Un nom de variable ne contient pas d'espace et ne doit pas débuter par un chiffre. Il n'est pas recommandé de le faire débuter par le caractère «_» .

Affectation de variables

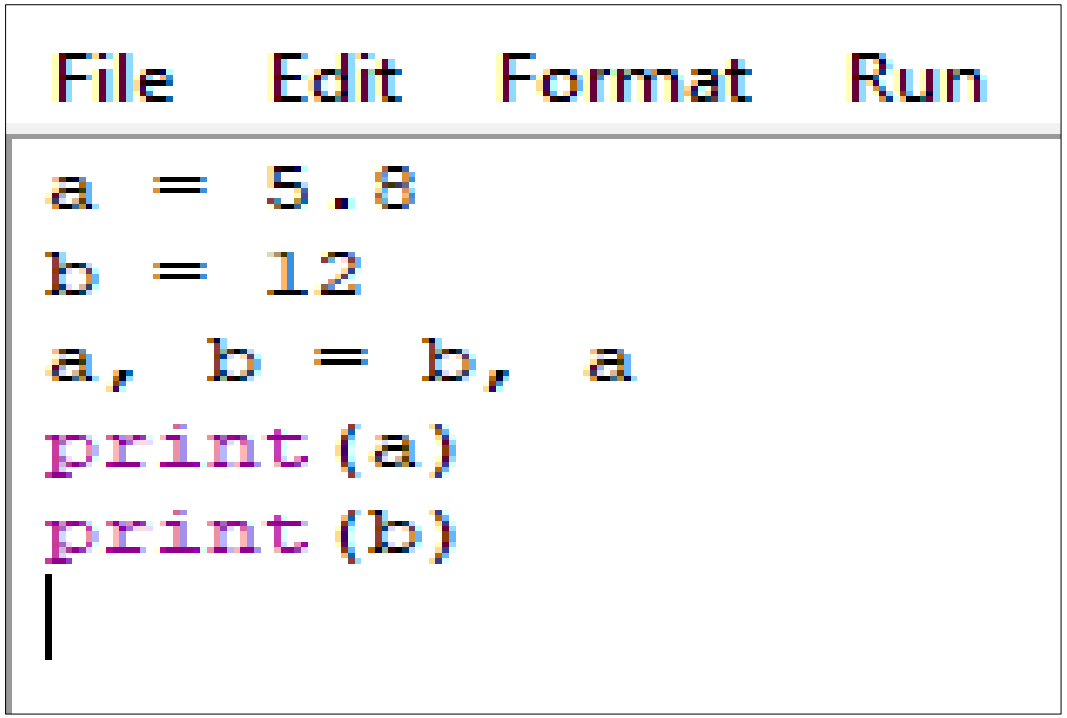
- L'instruction **x = 3** signifie qu'on attribue la valeur située à droite de l'opérateur « = » à la variable située à gauche.
- Dans l'instruction x = y – 3
  1. l'opération « y – 3 » est d'abord évaluée et ensuite
  2. le résultat de cette opération est affecté à la variable x
- **Affectation parallèle**:

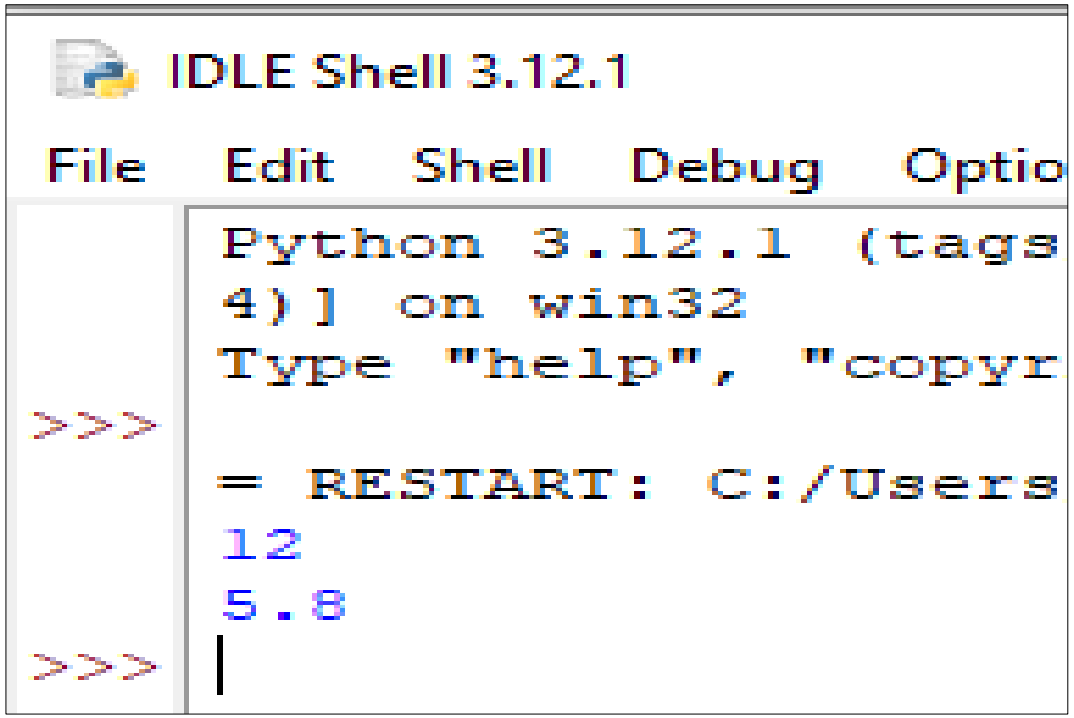
  x=8; y=8; z=12 est l'équivalent à **x, y, z=8, 8, 12**

  print(x, y, z)

Affectation de variables

- Exemple:





Mots réservés

```
>>> help()
```

```
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

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

help>
```

Remarque : il faut absolument éviter d'utiliser un **mot réservé** par Python comme nom de variable (par exemple : print, range, for, from, etc.).

Les types de variables

- Les variables de type immuable ne peuvent pas êtres modifiées:
  - **bool**: booléen (True ou False)
  - ***int****: entier*
  - ***float****: réel (le point est le séparateur décimal*)
  - **complex**: Complexe ex: 3+4j , 4j
  - ***str****: chaine de caractères reconnue par ‘ ‘ ou “ “*
  - **tuple**: séquence d’éléments de type identiques ou différents
- Les variables de type mutables peuvent être modifiées
  - **list**: liste d’éléments de type variable
  - **set**: ensemble: collection non ordonnée d’éléments uniques
  - **dict**: dictionnaire :structure non ordonnée d’éléments de type clé-valeur

Type de variables

- **type(var)** renvoie le type de la variable en argument **var.**
- **id(var)** renvoie un entier, représentant **<u>l'identifiant interne</u>** de la variable en argument **var** quel que soit son type.

```
>>> x=3
>>> type(x)
<class 'int'>
>>> id(x)
140711647853048
>>>
```

```
# Déclaration d'une variable x de type booléen
x=False
print(type(x))
print(id(x))
```

```
<class 'bool'>
140730135869576
```

Opérateurs

Les opérateurs arithmétiques		Les opérateurs de comparaison	
x+y	Addition	**x<y**	Strictement inférieur
x-y	soustraction	**x<=y**	Inférieur ou égal
x*y	Multiplication	**x>y**	Strictement supérieur
x/y	Division réelle	**x>=y**	Supérieur ou égal
xn**	Élévation à la puissance	**x==y**	Égal
x//y	Division entière	**x!=y**	Différent
x%y	Reste de division entière	**x is y**	x et y représentent le même objet
-x	Opposé - opérateur unaire	**x is not y**	x et y ne représentent pas le même objet

Exemples

- Opérations arithmétiques de base:

```
>>> x = 45
>>> x + 2
47
>>> x - 2
43
>>> x * 3
135
>>> y = 2.5
>>> x - y
42.5
>>> (x * 10) + y
452.5
```

Remarque:

- ✓ Si vous mélangez les types entiers et floats, **le résultat est renvoyé comme un float** (car ce type est plus général).
- ✓ L'utilisation de parenthèses permet de gérer **les priorités**.

Exemples

- L'opérateur **/** effectue une division. Contrairement aux opérateurs **+, -** et *****, celui-ci **renvoie systématiquement un float**:

```
>>> 3/2
1.5
>>> 4/2
2.0
```

- L'opérateur **puissance** utilise les symboles ****** :

```
>>>  2**3
8
>>>  2**4
16
```

- Pour obtenir **le quotient** et **le reste d'une division entière**, on utilise respectivement les symboles **//** et **modulo %:**

```
>>> 5 // 4
1
>>> 5 % 4
1
>>> 8 // 4
2
>>> 8 % 4
0
```

Opérateurs combinés

- Il existe des opérateurs « **combinés** » qui effectue une opération et une affectation en une seule étape :

```
>>> i = 0
>>> i = i + 1
>>> i
1
>>> i += 1
>>> i
2
>>> i += 2
>>> i
4
```

- L'opérateur **+=** effectue une addition puis affecte le résultat à la même variable.
  - Cette opération s'appelle une « **incrémentation** ».
  - Les opérateurs **-=**, ***=** et **/=** se comportent de manière similaire pour la soustraction, la multiplication et la division.

Priorités des opérateurs

****	puissance
***, */*, *//*, %	multiplication, division, division entière, reste
+, -	addition et soustraction
&, ^, |	Et bit à bit, Ou Exclusif bit à bit , Ou bit à bit
is, is not	Tests d'appartenance
<, <=, >, >=, !=, ==	Comparaisons
or , and, not x	Opérateurs logiques

Les priorités des opérateurs de la plus hausse à la plus basse

Exemples

- Exemple 1:

```
>>> 2**3*2
16
>>> 2*3**2
18
```

- Exemple 2:

```
>>> x=2
>>> y=3
>>> x&y
2
>>> x^y
1
>>> x|y
3
>>>
```

```
>>> a=5
>>> b=1
>>> (a==5) & (b<2)
True
```

- Exemple 3:

```
>>> x=2
>>> y=3
>>> x and y
3
>>> (x>3) and (y==3)
False
>>> x or y
2
>>> (x>3) or (y==3)
True
>>> not x
False
>>>
```

Opérations sur les chaines de caractères

- Pour les chaînes de caractères, deux opérations sont possibles, l'addition (**concaténation**) et la multiplication (**concaténation multiple**) :

```
>>> chaine = "Salut"
>>> chaine
'Salut'
>>> chaine + " Python"
'Salut Python'
>>> chaine * 3
'SalutSalutSalut'
```

- L'opérateur d'addition **+ concatène (assemble)** deux chaînes de caractères.
- L'opérateur de multiplication ***** entre un nombre entier et une chaîne de caractères duplique (répète) plusieurs fois une chaîne de caractères.

Conversion de types

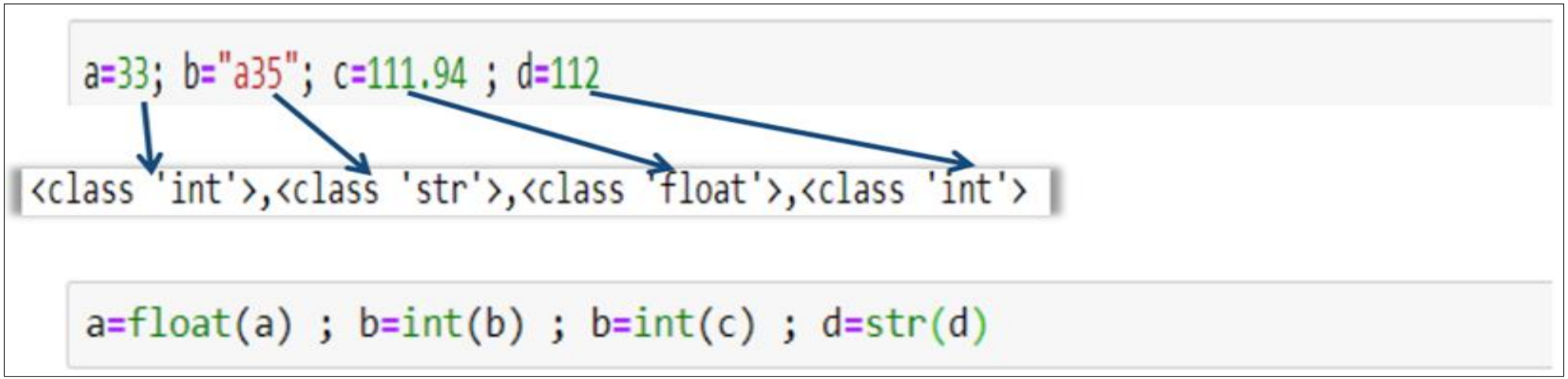
```
>>> ch="12"
>>> print(type(ch))
<class 'str'>
>>> x=ch+2
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    x=ch+2
TypeError: can only concatenate str (not "int") to str
>>>
```

```
>>> ch="12"
>>> type(ch)
<class 'str'>
>>> x=int(ch)+2
>>> x
14
```

- Python propose la conversion de types, c'est-à-dire passer d'un type numérique à une chaîne de caractères ou vice-versa avec les fonctions int(), float() et str().

Conversion de types

- Règles à suivre lors de la conversion des types
  - Entier (int) → réel (float) : float()
  - Réel (float)→Entier(int): int()
  - Entier (int) , réel (float) → chaine de caractères : str()




```
>>> a=33
>>> a=float(a)
>>> a
33.0
>>>
```

```
>>> b="a35"
>>> b=int(b)
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    b=int(b)
ValueError: invalid literal for int() with base 10: 'a35'
```

```
>>> c=111.64
>>> c=int(c)
>>> c
111
```

```
>>> d=112
>>> d=str(d)
>>> d
'112'
```

Exemples

- Chaine de caractères (str) → Entier (int) : int(‘’chaine’’)

```
annN=input("année de naissance")
age=2023-int(annN)
print("age : ",age)
print(type(age))
```

```
année de naissance1991
age :  32
<class 'int'>
```

- Chaine de caractères str → type adéquat : eval(‘’ chaine’’)

```
age=eval(input("Age = "))
print("la valeur saisie d'age =", age)
print(type(age))
```

```
Age = 28
la valeur saisie d'age = 28
<class 'int'>
```

Input() et eval()

- La fonction **input()**:
  - ne requiert pas de paramètres, cependant, elle peut avoir un seul paramètre qui est le texte affiché avant la saisie de l'utilisateur,
  - la valeur retournée par la fonction input() est le texte saisi par l'utilisateur sous la forme d'une chaîne.
- La fonction **eval()** :
  - évalue le contenu de la chaine fournie en argument comme étant une expression Python dont elle doit renvoyer le résultat.
  - Par exemple : eval("7 + 5") renvoie l'entier 12

Exemples

- Exemple 1:

```
>>> a=eval("12.5")
>>> a
12.5
>>> type(a)
<class 'float'>
```

- Exemple 3:

```
>>> a=int("12.5")
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    a=int("12.5")
ValueError: invalid literal for int() with base 10: '12.5'
```

- Exemple 2:

```
ch="12.4"
print(type(ch))
x=eval(ch)+2
print(x,type(x))
```

```
<class 'str'>
14.4 <class 'float'>
```

Affichage des variables : la fonction print()

- Print() affiche l'argument qu'on lui passe entre parenthèses et un retour à ligne → instruction de sortie
- Ajoute un retour à la ligne par défaut
- Si on ne veut pas afficher ce retour à la ligne, on peut utiliser l'argument **end**:

```
>>> print("Bonjour");print("le monde")
Bonjour
le monde
>>> print("Bonjour", end="");print("le monde");
Bonjourle monde
>>> print("Bonjour", end=" ");print("le monde");
Bonjour le monde
>>>
```

La fonction print()

- La fonction print() peut également afficher le contenu d'une variable quel que soit son type.

```
>>> x=3
>>> print(x)
3
```

- Il est également possible d'afficher le contenu de plusieurs variables (quel que soit leur type) en les séparant par des virgules :

```
>>> x=32
>>> nom="Ahmed"
>>> print(nom, "a", x, "ans")
Ahmed a 32 ans
>>>
```

La fonction print()

- la fonction print() possède l'argument par mot-clé sep qui met une chaine séparateur entre les valeurs des arguments :

```
>>> x=32
>>> nom="Ahmed"
>>> print(nom, "a", x, "ans", sep="")
Ahmeda32ans
>>> print(nom, "a", x, "ans", sep="-")
Ahmed-a-32-ans
>>>
```

- Une autre manière pour afficher deux chaines de caractères l'une à coté de l'autre sans espace est de les concaténer en utilisant l'opérateur +.

```
>>> couleur="Noir"
>>> couleur1="Blanc"
>>> print(couleur, couleur1)
Noir Blanc
>>> print(couleur+couleur1)
NoirBlanc
>>> print(couleur, couleur1,sep="")
NoirBlanc
>>> print(couleur, couleur1,sep='')
NoirBlanc
>>> print(couleur, couleur1,sep=' ')
Noir Blanc
```

Écriture formatée

- L'écriture formatée est un mécanisme permettant d'afficher des variables **avec un certain format.**
- Depuis la version 3.6, Python a introduit les **f-strings** (formatted string Literals) pour insérer des variables dans les chaines de caractères et de les mettre en forme.

```
>>> x=39
>>> nom="Salah"
>>> print(f"{nom} a {x} ans")
Salah a 39 ans
>>>
```

- Il suffit de passer un nom de variable au sein de chaque couple d'accolades et Python les remplace par leur contenu.

Format des nombres

- f-strings implémente également une manière de mettre en forme les nombres.
- Exemple:

```
prixtotal=20
NbrArticles=3
prixUnitaire=prixtotal/NbrArticles
print ("le prix unitaire est égal:", prixUnitaire)
```

```
le prix unitaire est égal: 6.666666666666667
```

Trop de décimales

Format des nombres

- Pour formater un **float** et l'afficher avec deux puis trois décimales:

```
prixTotal=20
nombreArticles=3
prixUnitaire = (prixTotal / nombreArticles)
print ( f" Le prix unitaire d'articles est:  {prixUnitaire:.2f}")
print ( f" Le prix unitaire d'articles est:  {prixUnitaire:.3f}")
```

```
Le prix unitaire d'articles est:  6.67
Le prix unitaire d'articles est:  6.667
```

Syntaxe pour le formatage

print (f ' <texte> { <expression/variable> : <format> } <texte> ‘)

- Pour les **float**, le format se trouve sous cette forme:

[alignement][signe][largeur][groupage][.précision][type]

- **alignement** : détermine ou le nombre est aligné dans sa zone
  - “**>**” aligne à droite par défaut
  - “**<**” aligne à gauche
  - “**^**” centré
  - “**=**” aligne le signe à gauche et le nombre à droite

Syntaxe pour le formatage

- **Signe** : détermine l'affichage du signe
  - "+" indique que le signe + doit être affiché ainsi que le -
  - "-" indique que le signe - doit être affiché (par défaut)
  - " " n'affiche pas le + mais insère un espace à la place
- **largeur** : détermine la place qui doit être réservée pour l'affichage du nombre
- **Groupage** détermine le symbole de séparation tous les 3 chiffres
  - "_"
  - "'"
- **précision** : détermine le nombre de chiffres après la virgule
- **Type :** détermine le mode d'affichage
  - "e" ou "E" notation scientifique
  - "f" ou "F" affichage classique

Exemple

```
>>> nombre = 357568.12312
>>> nombre2 = 568.568768
>>> nombre3 = -34.3432
>>> nombre4 = 23
>>> print(f'{nombre : >+20_.4f} {nombre2 : >+20_.4f}')
>>> print(f'{nombre3 : >+20_.4f} {nombre4 : >+20_.4f}')

+357_568.1231            +568.5688
     -34.3432             +23.0000
```

Formatage des chaines

- Ce formatage est possible sur des chaînes de caractères avec la lettre **s** (comme string) :

```
>>> print("langage Python"); print("langage Java")
langage Python
langage Java
>>> print(f"langage{"Python":>10s}");print(f"langage {"Java":>4s}")
langage    Python
langage Java
```

Autres détails sur les f-strings

- Si on veut afficher des accolades littérales avec les f-strings, il faut les doubler pour échapper au formatage :

```
>>> print(f"Accolades littérales {{}} ou {{ ou }} et pour le formatage {10}")
Accolades littérales {} ou { ou } et pour le formatage 10
```

- Si on ne met pas de variable à formater entre les accolades dans une f-string, cela conduit à une erreur :

```
>>> print(f"accolades sans variable {}")
  File "<stdin>", line 1
SyntaxError: f-string: empty expression not allowed
```

Autres détails sur les f-strings

- Une fonctionnalité extrêmement puissante des f-strings est de supporter **des expressions** au sein des accolades.

```
>>> print(f"Le résultat de 5 * 5 vaut {5 * 5}")
Le résultat de 5 * 5 vaut 25
>>> print(f"Résultat d'une opération avec des floats : {(4.1 * 6.7)}")
Résultat d'une opération avec des floats : 27.47
>>> print(f"Le minimum est {min(1, -2, 4)}")
Le minimum est -2
>>> entier = 2
>>> print(f"Le type de {entier} est {type(entier)}")
Le type de 2 est <class 'int'>
```

- Pour les nombres très grands ou très petits, l'écriture formatée permet d'afficher un nombre en notation scientifique avec la lettre **e** :

```
>>> print(f"{1_000_000_000:e}")
1.000000e+09
>>> print(f"{0.000_000_001:e}")
1.000000e-09
```