

LINGE1225 : Programmation en économie et gestion

## Cours 1

Variables, opérateurs, types, fonctions prédéfinies

François Fouss & Marco Saerens

Année académique 2020-2021

## Livre de référence

- Chapitre 2 : Premiers pas
- Chapitre 5 : Principaux Types de données
- Chapitre 6 : Fonctions prédéfinies



## Plan

- Introduction
- Variable
- Types de données
- Opérateurs
- Fonctions prédéfinies
  - Bibliothèques de fonctions prédéfinies
  - Fonctions utiles
- Application

3

## Introduction

4

## Principe

- Les programmes
  - sont généralement écrits sur base d'algorithmes
  - et suivent donc généralement un processus contenant 3 étapes:
    1. Les données d'entrée (**input**) sont reçues (ce sont les données que l'on donne au programme)
    2. Certaines **opérations** sont effectuées sur ces données
    3. Le résultat (**output**) est produit

Exemple d'un programme qui fait l'addition de deux nombres (comme une calculatrice) :

- 1) Nous donnons comme **input** : 2 et 3.
- 2) Le programme fait comme **opération** l'addition des données (input).
- 3) Le programme nous donne comme résultat (**output**) : 5

5

## Définition

- Un langage de programmation est composé :
  - d'**instructions** (exécutées par l'ordinateur);
  - d'une **syntaxe** (règles qui précisent comment les mots et symboles peuvent être rassemblés pour former des instructions valides).
- Programmation :
  - fournir à l'ordinateur des instructions qu'il devra exécuter
- Programme :
  - fichier texte (sans aucune mise en page ou attribut de style)
  - où l'on écrit code source (le code que vous avez écrit dans un langage de programmation)

6

## Traduction

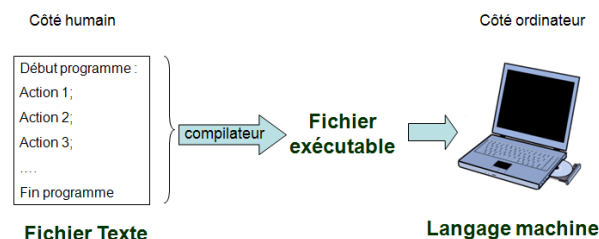
- Le langage est **traduit** en une suite d'instructions **codées** en langage machine
  - Quelles que soient les apparences, l'ordinateur ne parle pas le même langage que l'utilisateur.
- Nécessité d'un puissant **système de traduction** qui transpose les informations que nous lui fournissons dans sa langue natale

→ Lorsque vous avez écrit votre programme (code source) et que vous demandez à l'ordinateur de l'exécuter, il va premièrement le traduire dans sa propre langue.  
→ Dès cet instant, si vous avez mal écrit votre code source, l'ordinateur vous le fera savoir en disant qu'il ne le comprend pas.  
→ On peut apparenter cela comme traduire du français à l'anglais sauf que l'ordinateur ne comprend que s'il n'y a aucune faute de syntaxe (donc aucune faute d'orthographe).

7

## Compilateur

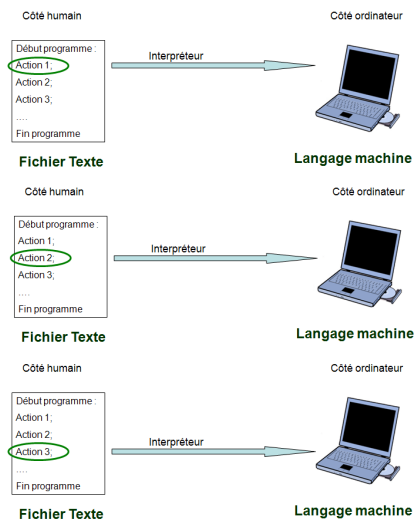
- Traduit l'**intégralité** d'un code source en un code **exécutable** (code dans la « langue de l'ordinateur »)
- Le résultat est conservé sous la forme d'un **fichier**
- Ce fichier peut être utilisé sans plus faire **aucune référence** au code source



8

## Interpréteur

- Traduit le code source **au fur et à mesure** de l'exécution du programme
- Utilise le **code source** au moment de l'exécution
- Le temps de traduction s'ajoute au temps d'exécution (cela ralentit le programme)



9

## Types d'erreur

- Erreurs de **syntaxe** (compile-time errors, détectées par le compilateur)
  - Attention à la **casse** et à la **punctuation**!
  - Cette erreur arrive lorsque l'ordinateur n'arrive pas à comprendre votre code source dû à une erreur de syntaxe.
- Erreurs à l'**exécution** (run-time errors)
  - Exemple: division par **0**
  - Liées au mécanisme d'**exception**
- Erreurs **logiques** (logical errors) ou **sémantiques**
  - Exemple: formule incorrecte, raisonnement incorrect.

10

## Compétences d'un programmeur

- Imaginer des solutions **innovantes** et **efficaces**
- Être capable d'exprimer ces solutions de manière **claire** et **complète**
- Apprendre à **déboguer** (càd : enlevé les « bugs » d'un programme qui le rend incorrect)
- **Commenter** son code (pour en faciliter la compréhension)
- Programmation
  - *Expliquer en détail à une machine ce qu'elle doit faire, en sachant d'emblée qu'elle ne peut pas véritablement « comprendre » un langage humain, mais seulement effectuer un traitement automatique sur des séquences de caractères.*

11

## Exemple

- Considérons une suite de nombres fournis dans le désordre
  - 42, 15, 3, 6, 9, 33, 1, 16, 22
- Comment expliquer à l'ordinateur à les remettre dans l'ordre croissant ?

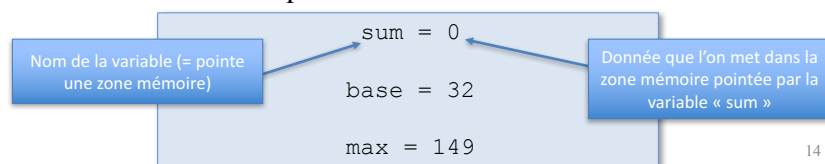
12

# Variables

13

## Variables

- Une **variable** est un nom (une référence) pour une localisation dans la mémoire (qui peut contenir une donnée)  
Une **variable** = une zone de **mémoire**
- Une variable doit être déclarée en spécifiant le nom de la variable
- Une variable appartient à un seul type – il existe différents types possibles (exemple ci-dessous: entier)
- Une variable peut se faire attribuer une valeur initiale grâce à la déclaration telle que



14

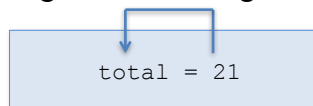
## Nom de variable

- Courts et explicites
- Séquence de lettres et de chiffres qui doit toujours commencer par une lettre
- Sans lettres accentuées, cédilles, espaces ou caractères spéciaux (excepté \_)
- La casse est significative
- Convention: tout en minuscule

15

## Affectation

- Opération par laquelle on établit un lien entre le nom de la variable et sa valeur (son contenu)
- La déclaration d'assignement change la valeur d'une variable
- L'opérateur d'assignation est le signe =



total = 21

- L'expression sur la droite est évaluée et le résultat est stocké dans la variable sur la gauche
- Le symbole = n'a donc pas la même signification qu'en mathématiques
- La valeur qui était dans `total` est écrasée et remplacée par la nouvelle valeur
- Il est possible d'affecter seulement une valeur à une variable

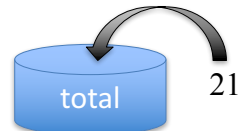
16



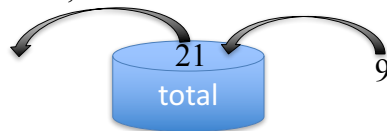
## Affectation

On peut voir une variable comme une boîte qui contient une donnée :

Lorsque l'on fait :  $\text{total} = 21$ , on met la valeur 21 dans la boîte qui porte le nom « total »



Si on fait par la suite :  $\text{total} = 9$ , on enlève de la boîte « 21 » et on met « 9 » à la place



➤ « 21 » n'est plus dans la boîte « total », la donnée est « écrasée » et remplacée par la nouvelle valeur : 9.

17

## Affectations multiples

**$x = y = 7$**

est un raccourci d'écriture pour

$x = 7$

$y = 7$

**$a, b = 4, 8.33$**

est un raccourci d'écriture pour

$a = 4$

$b = 8.33$

18

## Affectation parallèle

**a, b = b, a**

est un raccourci d'écriture pour

int = a

a = b

b = int

Et permet donc d'échanger les valeurs contenues dans a et b

19

## Variables

- Définir une variable et lui attribuer/affecter une valeur (avec le signe égal)

- Afficher: de deux manières différentes :

1) Entrer au clavier le nom de la variable, puis <Enter>

2) Utiliser la **fonction prédéfinie**

**print()**

Rq: print affiche la variable telle qu'elle a été encodée. L'autre méthode affiche des apostrophes car le type de la variable = chaîne de caractères.

Le signe « # » veut dire que l'on fait un commentaire, le compilateur n'en tiendra pas compte.

### #Affectation

```
>>> n = 7
```

```
>>> msg = "Quoi de neuf?"
```

### #Afficher la valeur

```
>>> n #Appeler la variable  
7
```

```
>>> msg  
'Quoi de neuf?'
```

```
>>> print(msg) #Commande print()  
Quoi de neuf?
```

20

## Mots réservés

- En python, il existe 33 « mots réservés » qui ne peuvent pas être utilisés comme nom de variable car ils sont utilisés par le langage lui-même.

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

21

## Commentaires

- Commencent toujours par le caractère #
  - Et s'étendent jusqu'à la fin de la ligne courante
- Ignorés par le compilateur
- Particulièrement utiles pour la lecture et la compréhension d'un programme
- Les espaces placés à l'intérieur des instructions et expressions sont également presque toujours ignorés (sauf s'ils font partie d'une chaîne de caractères)

Exemple :

```
1 x = 7 # Affectation de la variable x
2 msg = "Quoi de neuf ?" # la variable msg est
3                        # de type "string"
4
5 print (x) # Instruction pour afficher le contenu
6           # de la variable x
```

/!\ le symbole « # » indique que tout ce qui se situe à droite sur la même ligne est un commentaire.  
Pour faire un commentaire sur plusieurs lignes, il faut remettre à chaque fois le symbole « # »

22

# Types de données

23

## Types de données

- Toute variable appartient à un **type**.
- Un **type de donnée** =
  - un ensemble de valeurs appartenant au type : le domaine
  - un ensemble d'opérations applicables aux valeurs du type

```
type(25)  
<class 'int'>
```

Les **entiers**

```
type("bonjour")  
<class 'str'>
```

Les **strings** (chaîne de caractères)

```
type(24.2)  
<class 'float'>
```

Les **réels** (réels en nombre à virgule)

```
type(True)  
<class 'bool'>
```

Les **booléens** (vrai/faux)

24

## Types de données

➤ Il existe différents types de données en Python :

Types	Exemple
Entiers	5, 6, 876, 21355,...
Strings	'abc', 'd', 'e', 'fghij',...
Booléen	False, True
Réels	34.5 224.0 345.7890987 , ...

*Remarque :*

En Python, le fait d'assigner une valeur à une variable, cette variable est automatiquement créée avec le type qui correspond au mieux à la valeur fournie:

**Python est typé dynamiquement**

25

## Typage dynamique vs statique

Pour comprendre la différence entre du typage dynamique (ex : Python) et statique (ex : Java), il faut comprendre : *Pourquoi il existe différents types de données ?*

Il faut savoir qu'un ordinateur utilise le binaire pour stocker des informations, c'est-à-dire qu'une donnée est encodée sous forme d'une séquence de 1 et de 0 (des bits).

Par exemple, le chiffre 3 vaut 11 en binaire, le chiffre 9 vaut 1001 et 57 vaut 111001 où 3, 9 et 57 sont des entiers.

On voit qu'il y a un problème : si l'ordinateur reçoit : 111001, est-ce qu'il doit lire 57 ou 3 suivis de 9 ? Pour y résoudre, il faut dire à l'ordinateur, sur combien de bits est représenté un entier.

Disons que l'ordinateur doit lire les entiers sur 8 bits. C'est-à-dire que tous les 8 bits, il sait qu'il a un nouvel entier. Sur 8 bits : 3 vaut 00000011, 9 vaut 00001001 et 57 vaut 00111001.

→ Ici, il ne confondra plus 00111001, et 0000001100001001.

57                      3                      9

*Maintenant que vous avez compris ceci, pourquoi différents types ?*

26

## Typage dynamique vs statique

### Pourquoi différents types ?

Pour un traitement correct des données et une bonne utilisation de la mémoire

Premièrement, il faut que l'ordinateur sache à quel type de données il a affaire. En effet, le symbole « + » n'a pas la même signification si l'on se trouve avec des integers (la somme) ou des strings (la concaténation). Chaque type de données a des méthodes qui lui est propre. Donc l'ordinateur doit savoir en face de quel type il est (on parlera d'objet, on verra ce que c'est au dernier chapitre).

Deuxièmement, les types sont représentés sur un nombre différent de bits. Certains en ont par convention minimum 32 (ex : les floats), d'autres minimums 64 (les doubles). Par conséquent, un double prend plus de mémoire pour être stocké, il serait donc inutile de consacrer 64 bits à un float qui n'en a besoin que de 32 ! C'est de la mémoire perdue pour d'autres utilisations.

### Quelle est la différence entre un langage de type dynamique et statique ?

Un langage dynamique (ex : Python) n'a pas besoin qu'on lui dise de qu'elle type est la variable (qui contient la donnée), il le fait tout seul lorsque vous donnez une valeur à la variable. De plus, si vous changez de type de données pour une même variable, il va automatiquement changer le nombre de bits nécessaires pour stocker la donnée dans la variable.

Un langage statique (ex : Java) a besoin que l'on lui précise d'avance le type de la variable et ce type ne peut plus jamais changer pour une même variable !

En Java, on définit le type de la variable devant la variable

```
1 i = 5
2 f = 3.14
3
4 int i = 5
5 float f = 3.14
```

Python

Java

27

## Types de données

➤ Interroger une variable à propos de son type ?

➤ Utiliser la **fonction prédéfinie** `type(<variable>)`

➤ Créer : int, float, boolean, string

Vous n'avez pas besoin de définir le type des données, Python va le définir tout seul.

Exemple :

En bleu = la réponse de l'ordinateur

```
x=2
type(x)
<class 'int'>

x=False
type(False)
<class 'bool'>

my_int = 3
my_float = 1.24
my_bool = True
My_string = "ok"
```

28

## Les réels (float)

Pour représenter un réel, il faut obligatoirement un « . » suivi des chiffres après la virgule même si c'est « 0 ». Au sinon, c'est considéré comme un entier (voir dia suivante)  
Donc 0 est différent de 0.0 en terme de TYPAGE (ce ne sera pas le même nombre de bits consacré) !

➤ Exemple de valeurs : 0.0, 2.0, 0.04, ..., 7e8, 9.07e-23, ...

➤ Représentés en nombre à virgule

- Compris entre  $10^{-308}$  et  $10^{308}$
- 12 chiffres significatifs

➤ Opérations :

x = 0.5

3.0 \* x \*\* 2 + 4.0 \* x - 2.0

Rq : priorités des opérations

29

## Les entiers (integer)

➤ Exemple de valeurs : 0, 1, 2, ..., -1, -2, ...

➤ Illimités : 10 \*\* 1000

➤ Opérations :

x=2

3 \* x \*\* 2 + 4 \* x - 2

(42 // 5) \* 5 + 42 % 5

Division entière

\* = fois

\*\* = exposant

% = reste de la division

// = diviser

30

## Chaîne de caractères (string)

- Suite de caractères délimités par des apostrophes ou des guillemets
- Exemple de valeurs : "Bonjour", "Salut", "il dit 'non' ", "m".
- Opérations :

"2" + "2"                      "22"  
"progra" + "mmation"      "programmation"

Concaténation  
PAS addition !

concaténation = assembler

31

## Chaîne de caractère : indijage

- Les chaînes de caractères sont des séquences de caractères (char).

0	1	2	3	4	5	6
↓	↓	↓	↓	↓	↓	↓
"b	o	n	j	o	u	r"
↑						↑
caractère						caractère

- Les éléments d'une séquence sont **indijés** à partir de zéro.
- Pour extraire un caractère d'une chaîne : on accole au nom de la variable de la chaîne, son indice entre crochets

Exemple :

```
#Chaîne de caractère  
nom = "Cédric"  
print(nom[1], nom[3], nom[5])  
  
é r c
```

32



## Extraction de fragments de chaînes

- Comment extraire une petite chaîne d'une chaîne plus longue?
- Indiquer entre crochets les indices correspondant au début et à la fin de la « tranche » que l'on souhaite extraire
- *Rmq* : dans la tranche [n,m], le nième caractère est inclus mais pas le mième :

```
#chaîne de caractère : extract.  
ch = "Juliette"  
print(ch[0:3])  
Jul
```

- Moyen de mémorisation : se représenter les indices pointant des emplacements situés *entre* les caractères comme ci-dessous :

```
c h = " J u l i e t t e "  
      ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑  
      0 1 2 3 4 5 6 7 8
```

Avec cette exemple : ch[3:7] extraira « iett ».

33

## Chaîne de caractères: Immutabilité

- Le contenu d'une chaîne de caractères existante ne peut être modifié.
- Cela implique qu'il n'est pas possible d'utiliser l'opérateur [] dans la partie gauche d'une instruction d'affectation.

```
#chaîne de caractère : non modifiable
```

```
salut = "bonjour à tous"  
salut[0] = "B" ← Essai de modification de la première lettre de la  
print(salut)      chaîne de caractère
```

- ERREUR : le résultat attendu serait 'Bonjour à tous' or le scripte détecte une erreur. On essaie de remplacer un caractère par un autre et cela n'est pas permis

34

## Booléens

- Une valeur booléenne représente une condition Vraie ou Fausse
- Un booléen peut aussi être utilisé pour représenter deux états quelconques, comme une ampoule allumée ou éteinte
- Les mots True et False sont les seules valeurs valides pour le type booléen

35

## Conversion de types

- `int(x)` convertit x en valeur **entière**
- `float(x)` convertit x en valeur **réelle**
- `str(x)` convertit x en **chaîne de caractère**

### #Conversion de types

#### # int(x)

```
int(3.14)      3
int(3.999)     3
int("42")      42
int("15cm")    ERREUR
```

"cm" ne sait pas être convertie en un entier

#### # float(x)

```
float(34)      34.0
```

#### # str(x)

```
str(3.14)      "3.14"
```

36

# Opérateurs

37

## Opérateurs arithmétiques

- Une *expression* est une combinaison de un ou de plusieurs opérands et leur opérateur.
- Une *expression arithmétique* calcule le résultat numérique et utilise des opérateurs arithmétiques
- NB: le séparateur décimal est toujours le . et pas la ,

Addition	+
Soustraction	—
Multiplication	*
Division	/
Division entière	//
Modulo	%
Puissance	**

38

## Division & modulo

- Division entière: //  
le résultat sera un 'integer'  
sauf si au moins l'un des  
chiffres est un float
- Le modulo (%) donne le  
reste après avoir divisé le  
deuxième opérant par le  
premier

### #Opérations basiques

#### # division

```
14 // 3      #equals 4
8 // 12      #equals 0
3.6 // 2     #equals 1.0
```

#### # modulo

```
14 % 3      #equals 2
8 % 12      #equals 8
```

39

## Division & modulo

/\ Ne pas confondre division entier, division réelle et modulo !

### # division entière (//):

```
14 // 3      #equals 4
8 // 12      #equals 0
3.6 // 2     #equals 1.0
```

### # division réelle (/):

```
14 / 3       #equals 4.666666
8 / 12       #equals 0.666666
3.6 / 2      #equals 1.8
4 / 2        #equals 2.0
```

La division réelle donnera  
toujours un float (un réel)

### # modulo

```
14 % 3      #equals 2
8 % 12      #equals 8
3.6%2       #equals 1.6
```

40

## Ordre des opérations

- Les opérateurs peuvent être combinés dans des expressions complexes

```
result = 23 + 34 / 123 - 65
```

- Les opérateurs ont des priorités bien définies qui déterminent l'ordre dans lequel ils sont évalués
- Multiplication, division et le reste sont évalués en priorité par rapport à l'addition, soustraction et à la concaténation de string.
- Les opérateurs arithmétiques avec la même priorité sont évalués de gauche à droite.
- Les parenthèses peuvent être utilisées pour forcer l'ordre d'évaluation

41

## Priorité des opérations (PEMDAS)


1. P (parenthèses)
2. E (exposants)
3. M (multiplication) et D (division) + *modulo (en Python)*
4. A (addition) et S (soustraction)


- A priorité égale: de gauche à droite
- Utiliser les parenthèses ()


42


## Ordre des opérations

➤ Quel est l'ordre de l'évaluation dans les expressions suivantes ?

$$A + B + C + D + E$$


$$A + B * C - D / E$$



$$A / (B + C) - D \% E$$


$$A / (B * (C + (D - E)))$$



43

## Ordre des opérations


➤ Quel est l'ordre de l'évaluation dans les expressions suivantes ?

$$A + B + C + D + E$$



1 2 3 4

$$A + B * C - D / E$$


3 1 4 2

$$A / (B + C) - D \% E$$


2 1 4 3

$$A / (B * (C + (D - E)))$$


4 3 2 1

44

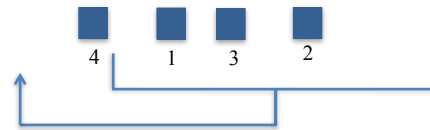
## Ordre des opérations

- L'ordre des opérations d'affectation a une plus petite priorité que les opérateurs arithmétiques

sum est une variable qui contient une valeur

Premièrement l'expression de droite est évaluée

answer = sum / 4 + 44 \* 217



Après le résultat est stocké dans la variable du côté gauche

45

## Opérations basiques

```
#Basic arithmetic operations  
# + - * / % **
```

```
#addition
```

```
x = 4  
y = 2  
x + y  
6
```

affectation

```
#multiplication
```

```
x * y  
8
```

```
#puissance
```

```
x ** 2  
16
```

46

## Affectation abrégée

➤ Il y a plusieurs façons d'affecter une valeur à une variable telle que :

Opérateur	Exemple	Équivalent à
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Exemple :

```
>>> x = 1
>>> x += 5
>>> x
6

>>> x = 2
>>> y = 3
>>> x *= y
>>> x
6
```

47

## Autres opérateurs :

Ces deux opérateurs seront  
vues plus en profondeur  
dans le prochain chapitre

### Opérateurs logiques

NOT	Logical
AND	Logical
OR	Logical

### Opérateurs de comparaison (booléens)

==	égal à
!=	pas égal à
<	plus petit que
>	plus grand que
<=	plus petit ou égal que
>=	plus grand ou égal que

48



# Fonctions prédéfinies

## Bibliothèques de fonctions prédéfinies

49

## Importer un module de fonctions

Nous avons vu des fonctions intégrées au langage Python telles que `print()`, `type()`, ...

Cependant les fonctions intégrées au langage sont relativement peu nombreuses. Les autres sont regroupées dans des fichiers séparés que l'on appelle des **modules**.

- **Modules** = fichiers qui regroupent des ensembles de fonctions prédéfinies.
- **Bibliothèques** = module qui regroupe des ensembles de fonctions apparentées

*Syntaxe :*

```
from <module> import <fonction>
```

50

## Importer un module de fonctions

*Exemple :*

Le module `math`, contient des définitions de nombreuses fonctions mathématiques telles que sinus, cosinus, tangente, racine carrée etc.

```
from math import *
```

Importer toutes les fonctions

nom du module

- Interprétation : Il faut inclure dans le programme courant toutes les fonctions signe `*` du module `math`, lequel contient une bibliothèque de fonctions mathématiques préprogrammées.

51

## Module math

Sans import le module `math`, le programme nous renverrait une erreur disant qu'il ne connaît pas « `sqrt()` »

**#Utilisation des fonctions du module <math>**

```
from math import *
```

```
nombre = 121
```

```
angle = pi/6
```

```
print('racine carrée de', nombre, '=', sqrt(nombre))
```

```
print('sinus de', angle, 'radians', '=', sin(angle))
```

Utilisation de  
fonction du  
module `math`

Ce qui donne :

```
racine carrée de 121 = 11.0
```

```
sinus de 0.523598775598 radians = 0.5
```

Utilisation de  
fonction du  
module `math`

52

## Caractéristiques des fonctions

- Une fonction apparaît sous la forme d'un nom quelconque associé à des parenthèses  
exemple : `sqrt()`
- Dans les parenthèses, on transmet à la fonction un ou plusieurs arguments  
exemple : `sqrt(121)`
- La fonction fournit une valeur de retour (on dira aussi qu'elle « retourne », ou mieux, qu'elle « renvoie » une valeur)  
exemple : 11.0

53

## Conseils

- `<math>` est une toute petite partie des modules de Python.
- N'hésitez pas à regarder dans la documentation bibliothèque de Python les très nombreuses différentes fonctions disponibles.
- ! Vous ne pourrez utiliser que certains modules pour l'examen

54

## Random

- Programmes déterministes = programmes qui feront toujours la même chose chaque fois qu'on les exécute.
- Dans son module `random`, Python propose toute une série de fonctions permettant de générer des nombres aléatoires qui suivent différentes distributions mathématiques.

Exemple

```
from random import *
```

55

## Random

```
from random import *  
def list_aleat(n):  
    s = [0]*n  
    for i in range(n):  
        s[i] = random.random()  
    return s
```

« n » est la valeur de l'argument passé à la fonction. Cette valeur est déterminée lors de l'appel à la fonction.

La fonction `random` permet de créer une liste de nombres réels aléatoires, de valeur comprise entre zéro et 1

```
list_aleat(3)  
[0.3322011398880431, 0.5642475883164183, 0.591966613411697]  
  
list_aleat(3)  
[0.941262175783142, 0.16765659221691698, 0.9420901408904373]
```

- Construction d'une liste de zéros de taille `n` et ensuite remplacement des zéros par des nombres aléatoires

56

## Exercice

- Créez un programme qui simule le lancer de deux dés, faites la somme et affichez le résultat de cette somme

57

## Solution

On spécifie au programme d'où vient « randint ». Pour ce faire, on fait : `<module>.<fonction()>` où le module a été importé préalablement.

```
import random
des1 = random.randint(1,6)
des2 = random.randint(1,6)
résultat = des1 + des2
print(résultat)
```

58

# Fonctions prédéfinies

## Fonctions utiles

59

## input()

input() fonction

- Permet d'interagir avec l'utilisateur
- L'utilisateur est invité à entrer des caractères au clavier et à terminer avec <Enter>.
- Lorsque cette touche est enfoncée, l'exécution du programme se poursuit, et la fonction fournit en retour une chaîne de caractères correspondant à ce que l'utilisateur a saisi.

```
#Fonction input()

prenom = input("Entrez votre
prénom :")

print("Bonjour," prenom)

print("Veuillez entrer un
nombre positif quelconque :",
end=" ")

ch = input()
nn = int(ch) #conversion chaine
           en nbre entier

print("Le carré de", n, "vaut",
n**2)
```

60

## type()

type() fonction

- Permet d'afficher le type de la variable entrée en paramètre

```
#Fonction type()

type(25)
<class 'int'>

type("bonjour")
<class 'str'>

type(24.2)
<class 'float'>

type(True)
<class 'bool'>
```

61

## len()

len(parameter) fonction

- Calcule le nombre d'éléments de l'argument `parameter` passé en paramètre
- Paramètre d'une fonction
  - Input nécessaire à l'exécution de la fonction
- Si `parameter` est une chaîne de caractères, `len(parameter)` renvoie le nombre de caractères de `parameter`

```
#Fonction len(parameter)

len("bonjour")
7

var1 = Coucou
len(var1)
6
```

62

## print()

print() fonction

- Permet d'afficher n'importe quel nombre de valeurs fournies en arguments
- Par défaut, ces valeurs seront séparées les unes des autres par un espace et se terminera avec un saut à la ligne.
- Possibilité de remplacer le séparateur par défaut par un caractère quelconque grâce à l'argument sep.

```
#Fonction print()

print("Bonjour", "à", "tous",
      sep="*")

Bonjour*à*tous

print("Bonjour", "à", "tous",
      sep=" ")

Bonjouràtous

print("Bonjour", "à", "tous")

Bonjour à tous
```

63

## range()

range(end)

- Renvoie une liste de nombres entiers allant de 0 à end-1

range(start, end)

- Renvoie une liste de nombres entiers allant de start à end-1

range(start, end, incr)

- Renvoie une liste de nombres entiers allant de start au plus grand nombre incrémenté inférieur à end-1, par incrément de incr

```
#Fonction range(end)

range(5)
Crée la liste [0,1,2,3,4]

#Fonction range(start,end)

range(0,10)
Crée la liste
[0,1,2,3,4,5,6,7,8,9]

#Fonction range(start,end,incr)

range(10,35,5)
Crée la liste
[10,15,20,25,30]
```

64



## abs()

`abs(nbre)` fonction

- Renvoie la valeur absolue de l'argument `nbre` passé en paramètre
- Si l'argument est un nombre entier ou flottant, le résultat est un nombre entier ou flottant

```
#Fonction abs(nbre)
```

```
abs(-3)  
3
```

```
var1 = -3.14  
abs(var1)  
3.14
```

65

## pow() et sqrt()

`pow(x, y)` fonction

- Renvoie la valeur de `x` élevée à la puissance `y`

`sqrt(x)` fonction

- Renvoie la racine carrée de `x`

```
#Fonction pow(x,y)
```

```
pow(2,3)  
8
```

```
#Fonction sqrt(x)
```

```
sqrt(16)  
4
```

66

# Application financière

67

## Intérêt simple & composé

- Un intérêt correspond à la rémunération d'un capital. Un intérêt peut être calculé:
  - pour un crédit, il sert alors à déterminer le coût du crédit
  - pour un placement, il sert alors à déterminer le rendement du placement

- Un **intérêt simple** est un intérêt qui est calculé uniquement sur le montant du capital, sans prendre en compte les intérêts antérieurs.

*exemple : un placement de 1000€ rémunère 2% d'intérêts simples annuels; au bout d'un an, ce placement dégage 20€ d'intérêts simples, au bout de deux ans, il dégagera toujours 20€*

- Les **intérêts composés** sont calculés sur le capital ainsi que sur les intérêts cumulés antérieurement

*exemple : un placement rémunère 2% d'intérêts simples. La première année il sera égal à celui de l'intérêt simple la deuxième année il sera égale à 1020€ + 2% de 1020€*

68

source : epargne.ooreka.fr

## Intérêt simple

Considérons un capital initial de  $C_0$  100 000 et un taux mensuel de 0,01.  
Quel est l'intérêt après 6 mois ? Quelle est la valeur acquise par le capital?

L'intérêt simple est donné par la relation :  $i_n = C_0 * n * i$

Le capital acquis est ensuite calculé par :  $C_n = C_0 + i_n = C_0 * (1 + n * i)$

Le capital acquis, mais en considérant des intérêts composés cette fois-ci, se calcule à partir de :  $C_n = C_0 * (1 + i)^n$



69

1. Quelles sont les variables importantes que je vais devoir définir dans mon programme ?

## Solution

Affectation des valeurs

Calcule intérêt + affichage valeur

Calcule capital + affichage valeur

### #Application 1

```
i = 0.01  
capital = 100000.0  
n = 6.0
```

### #interet

```
Is6 = i*capital*n  
print(Is6)
```

### #valeur acquise

```
Cs6 = capital + Is6  
print(Cs6)
```

```
Cc6 = capital*((1 + i)**n)  
Print(Cc6)
```

La solution pour un intérêt composé sera revisitée lorsque vous aurez vu les boucles

70