

Algorithmique de base

Module : Fondement de la programmation

Douglas Teodoro

Hes·so

Haute Ecole Spécialisée
de Suisse occidentale

Fachhochschule Westschweiz

University of Applied Sciences and Arts
Western Switzerland

2019-2020

SOMMAIRE

Objective

Rappel

La passage de paramètres

Passage de paramètres par valeur ou par référence

Portée des variables

OBJECTIVE

- ▶ Maîtriser le concept de portée d'une variable
- ▶ Maîtriser la passage de paramètres pour les fonctions et procédures
- ▶ Mettre en œuvre quelques structures pour résoudre des problèmes simples

SOMMAIRE

Objective

Rappel

La passage de paramètres

Passage de paramètres par valeur ou par référence

Portée des variables

RAPPEL

Les sous-programme

DÉCLARATION ET DÉFINITION

Fonction

```
1  """ nom
2  données: ...
3  résultat: ...
4  """
5  def nom_fonction(parametres) -> type_retour:
6      # bloc d'instructions
7      ...
8      return ...
9
10 # programme principal
11 ...
```

Procédure

```
1  """ nom
2  données: ...
3  résultat: ...
4  """
5  def nom_procedure(parametres):
6      # bloc d'instructions
7      ...
8      # pas d'instruction return
9
10 # programme principal
11 ...
```

- ▶ **def** : mot clé réservé pour indiquer la définition d'une fonction ou procédure
- ▶ **return** : mot clé réservé pour indiquer la valeur retourné par la fonction
- ▶ **nom** : nom donné au sous-programme, même convention que pour les variables
- ▶ **parametres** : la liste de paramètres passé au sous-programme
- ▶ **type_retour** : le type retourné (str, int, float, etc.)

DÉCLARATION ET DÉFINITION - EXEMPLE

```
1  """ algo: calcule puissance
2  données: n -> int, x -> float
3  résultat: puissance x^n -> float
4  """
5  def puissance(x: float, n: int) -> float:
6      resultat: float = 1
7      for cpt in range(1, n + 1):
8          resultat = resultat * x
9      return resultat
10
11 def print_res(x: float, n: int, y: float):
12     print("puissance de %.2f ^ %d est %.2f"
13           %(x, n, y))
14
15 x: float = float(input("x: "))
16 n: int = int(input("n: "))
17
18 resultat: float = puissance(x, n)
19 print_res(x, n, resultat)
```

- lignes 1 - 4 : documentation du programme

DÉCLARATION ET DÉFINITION - EXEMPLE

```
1  """ algo: calcule puissance
2  données: n -> int, x -> float
3  résultat: puissance x^n -> float
4  """
5  def puissance(x: float, n: int) -> float:
6      resultat: float = 1
7      for cpt in range(1, n + 1):
8          resultat = resultat * x
9      return resultat
10
11 def print_res(x: float, n: int, y: float):
12     print("puissance de %.2f ^ %d est %.2f"
13           %(x, n, y))
14
15 x: float = float(input("x: "))
16 n: int = int(input("n: "))
17
18 resultat: float = puissance(x, n)
19 print_res(x, n, resultat)
```

- lignes 1 - 4 : documentation du programme
- lignes 15 - 19 : programme principal

DÉCLARATION ET DÉFINITION - EXEMPLE

```
1  """ algo: calcule puissance
2  données: n -> int, x -> float
3  résultat: puissance x^n -> float
4  """
5  def puissance(x: float, n: int) -> float:
6      resultat: float = 1
7      for cpt in range(1, n + 1):
8          resultat = resultat * x
9      return resultat
10
11 def print_res(x: float, n: int, y: float):
12     print("puissance de %.2f ^ %d est %.2f"
13           %(x, n, y))
14
15 x: float = float(input("x: "))
16 n: int = int(input("n: "))
17
18 resultat: float = puissance(x, n)
19 print_res(x, n, resultat)
```

- lignes 1 - 4 : documentation du programme
- lignes 15 - 19 : programme principal
- lignes 5 - 9 : fonction qui calcule la puissance

DÉCLARATION ET DÉFINITION - EXEMPLE

```
1  """ algo: calcule puissance
2  données: n -> int, x -> float
3  résultat: puissance x^n -> float
4  """
5  def puissance(x: float, n: int) -> float:
6      resultat: float = 1
7      for cpt in range(1, n + 1):
8          resultat = resultat * x
9      return resultat
10
11 def print_res(x: float, n: int, y: float):
12     print("puissance de %.2f ^ %d est %.2f"
13           %(x, n, y))
14
15 x: float = float(input("x: "))
16 n: int = int(input("n: "))
17
18 resultat: float = puissance(x, n)
19 print_res(x, n, resultat)
```

- ▶ lignes 1 - 4 : documentation du programme
- ▶ lignes 15 - 19 : programme principal
- ▶ lignes 5 - 9 : fonction qui calcule la puissance
- ▶ lignes 11 - 13 : procédure qui affiche le résultat

SOMMAIRE

Objective

Rappel

La passage de paramètres

Passage de paramètres par valeur ou par référence

Portée des variables

LA PASSAGE DE PARAMÈTRES

DÉFINITION D'UNE FONCTION

Les 6 étapes de définition

nom : un **identificateur** suffisamment explicite

paramètres : la liste des paramètres d'**entrée-sortie** de l'algorithme

préconditions : une liste d'expressions booléennes qui précisent les **conditions d'application** de l'algorithme

appel : des **exemples d'utilisation** de l'algorithme avec les résultats attendus

description : une phrase qui dit **ce que fait** l'algorithme

code : la **séquence d'instructions** nécessaires à la résolution du problème

DÉFINITION D'UNE FONCTION - SYNTAXE

**nom de la
fonction**

**liste de
paramètres
formels**

**type de
la valeur
retournée**

```
1  def fonc_nom(p1: type, p2: type, ..., pn: type=valeur) -> type_retour:
2      """
3      description fonc
4      :param p1: desc p1
5      :param p2: desc p2
6      ...
7      :param pk: desc pk
8      :param pn: desc pn
9      :return: desc retour
10     """
11     # bloc d'instructions
12     ...
13
14     return ...
```

**documen-
tation**

instructions

**instruction
return**

LES PARAMÈTRES

- ▶ Quand en mathématiques on calcule la valeur d'une fonction $f(x)$, on lui demande de calculer la **valeur de la fonction** selon la **valeur de x**

$$\text{ex. : } f(x) = 2x + 1$$

- ▶ On **pass**e donc la valeur de x à la fonction f

$$\text{ex. : pour } x = 2 \rightarrow f(x) = 5$$

Passages de paramètres

Le **principe** est le même avec nos algorithmes : on **transmettre des valeurs** à nos procédures et fonctions

LES PARAMÈTRES

En informatique, il est possible de **passer comme paramètre** toute expression **retournant une valeur** :

Type de paramètres

- ▶ scalaire : ex. `10` → `range(10)`
- ▶ variable : ex. `var_x` → `print(var_x)`
- ▶ tableau : ex. `[1,2,3,4]` → `len([1,2,3,4])`
- ▶ fonction : ex. `fonc_x` → `print(len([1,2,3,4]))` ou `fonc_x(fonc_y)`

LES PARAMÈTRES - STRUCTURE

Pour les fonctions, les paramètres **se placent entre les parenthèses** situées après leur nom

```
1 """ nom
2 données: ...
3 résultat: ...
4 """
5 def fonc(paramètres) -> type_retour:
6     # bloc d'instructions
7     ...
8     return ...
9
10 # programme principal
11 ...
```

```
1 """ affiche parametres
2 données: p1 -> int, p2 -> int
3 résultat: somme p1 et p2
4 """
5 def somme(p1: int, p2: int) -> int:
6     """
7     somme p1 et p2
8     """
9     print("somme:", p1, p2)
10    r: int = p1 + p2
11    return r
12
13 # programme principal
14 res: int = somme(10, 20)
15 print("retour somme: %d" %res)
```

LES PARAMÈTRES - STRUCTURE

Pour les fonctions, les paramètres **se placent entre les parenthèses** situées après leur nom

```
1 """ nom
2 données: ...
3 résultat: ...
4 """
5 def fonc(parametres) -> type_retour:
6     # bloc d'instructions
7     ...
8     return ...
9
10 # programme principal
11 ...
```


```
1 """ affiche parametres
2 données: p1 -> int, p2 -> int
3 résultat: somme p1 et p2
4 """
5 def somme(p1: int, p2: int) -> int:
6     """
7     somme p1 et p2
8     """
9     print("somme:", p1, p2)
10    r: int = p1 + p2
11    return r
12
13 # programme principal
14 res: int = somme(10, 20)
15 print("retour somme: %d" %res)
```

LES PARAMÈTRES - STRUCTURE

Pour les fonctions, les paramètres **se placent entre les parenthèses** situées après leur nom

```
1 """ nom
2 données: ...
3 résultat: ...
4 """
5 def fonc(parameters) -> type_retour:
6     # bloc d'instructions
7     ...
8     return ...
9
10 # programme principal
11 ...
```

```
1 """ affiche parametres
2 données: p1 -> int, p2 -> int
3 résultat: somme p1 et p2
4 """
5 def somme(p1: int, p2: int) -> int:
6     """
7     somme p1 et p2
8     """
9     print("somme:", p1, p2)
10    r: int = p1 + p2
11    return r
12
13 # programme principal
14 res: int = somme(10, 20)
15 print("retour somme: %d" %res)
```

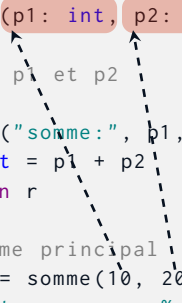
A dashed arrow originates from the arguments '10' and '20' in the function call 'somme(10, 20)' on line 14 and points to the parameters 'p1' and 'p2' in the function definition 'def somme(p1: int, p2: int)' on line 5.

LES PARAMÈTRES - STRUCTURE

Pour les fonctions, les paramètres **se placent entre les parenthèses** situées après leur nom

```
1 """ nom
2 données: ...
3 résultat: ...
4 """
5 def fonc(parameters) -> type_retour:
6     # bloc d'instructions
7     ...
8     return ...
9
10 # programme principal
11 ...
```

```
1 """ affiche parametres
2 données: p1 -> int, p2 -> int
3 résultat: somme p1 et p2
4 """
5 def somme(p1: int, p2: int) -> int:
6     """
7     somme p1 et p2
8     """
9     print("somme:", p1, p2)
10    r: int = p1 + p2
11    return r
12
13 # programme principal
14 res: int = somme(10, 20)
15 print("retour somme: %d" %res)
```



PASSAGE DE PARAMÈTRES - PORTÉE ET NOM

- Les **paramètres passés** à un sous-programme sont généralement des **variables locales** au programme ou sous-programme l'appelant

```
1  """algo: passage param
2  données: nb_char -> int: taille ligne
3  résultat: affiche une ligne de taille
           nb_char des étoiles
4  """
5  def affiche_ligne(n: int):
6      """
7      affiche ligne avec n étoiles
8      :param n: nombre de caractères
9      :return:
10     """
11     for i in range(1, n+1):
12         print("*", end=" ")
13
14 # données
15 nb_char: int = 10
16 affiche_ligne(nb_char)
```

PASSAGE DE PARAMÈTRES - PORTÉE ET NOM

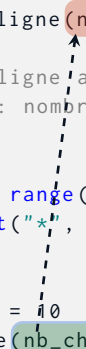
- ▶ Les **paramètres passés** à un sous-programme sont généralement des **variables locales** au programme ou sous-programme l'appelant
 - ▶ Ils **ne portent pas** forcément le **même nom**

```
1  """algo: passage param
2  données: nb_char -> int: taille ligne
3  résultat: affiche une ligne de taille
         nb_char des étoiles
4  """
5  def affiche_ligne(n: int):
6      """
7      affiche ligne avec n étoiles
8      :param n: nombre de caratères
9      :return:
10     """
11     for i in range(1, n+1):
12         print("*", end=" ")
13
14 # données
15 nb_char: int = 10
16 affiche_ligne(nb_char)
```

PASSAGE DE PARAMÈTRES - PORTÉE ET NOM

- ▶ Les **paramètres passés** à un sous-programme sont généralement des **variables locales** au programme ou sous-programme l'appelant
 - ▶ Ils **ne portent pas** forcément le **même nom**
- ▶ Ils sont **recupérés** au sein du sous-programme comme des **variables locales** au sous-programme

```
1  """algo: passage param
2  données: nb_char -> int: taille ligne
3  résultat: affiche une ligne de taille
         nb_char des étoiles
4  """
5  def affiche_ligne(n: int):
6      """
7      affiche ligne avec n étoiles
8      :param n: nombre de caractères
9      :return:
10     """
11     for i in range(1, n+1):
12         print("*", end=" ")
13
14 # données
15 nb_char: int = 10
16 affiche_ligne(nb_char)
```



A dashed arrow points from the `nb_char` argument in the function call `affiche_ligne(nb_char)` on line 16 to the `n: int` parameter in the function definition `def affiche_ligne(n: int):` on line 5. The parameter `n: int` is highlighted with a red box, and the argument `nb_char` is highlighted with a green box.

PASSAGE DE PARAMÈTRES - PARAMÈTRES PAR DÉFAUT

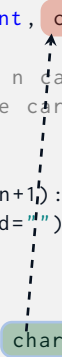
- On peut définir des **paramètres par défaut** en Python

```
1 """ algo: passage param
2 données: nb_char -> int: taille ligne
3 résultat: affiche une ligne de taille n
4 """
5 def affiche_ligne(n: int, char: str = "*"):
6     """
7     affiche ligne avec n caractères
8     :param n: nombre de caractères
9     :return:
10    """
11    for i in range(1, n+1):
12        print(char, end="")
13
14 # données
15 nb_char: int = 10
16 affiche_ligne(nb_char, char="#")
17 print()
18 affiche_ligne(nb_char)
```

PASSAGE DE PARAMÈTRES - PARAMÈTRES PAR DÉFAUT

- On peut définir des **paramètres par défaut** en Python

```
1 """ algo: passage param
2 données: nb_char -> int: taille ligne
3 résultat: affiche une ligne de taille n
4 """
5 def affiche_ligne(n: int, char: str = "*"):
6     """
7     affiche ligne avec n caractères
8     :param n: nombre de caractères
9     :return:
10    """
11    for i in range(1, n+1):
12        print(char, end=" ")
13
14 # données
15 nb_char: int = 10
16 affiche_ligne(nb_char, char="#")
17 print()
18 affiche_ligne(nb_char)
```



The diagram illustrates the flow of arguments in a function call. A dashed arrow points from the `char` argument in the function call `affiche_ligne(nb_char, char="#")` on line 16 to the `char` parameter in the function definition `def affiche_ligne(n: int, char: str = "*"):` on line 5. The parameter `char` in the definition is highlighted with a red box, and the argument `char="#"` in the call is highlighted with a green box.

PASSAGE DE PARAMÈTRES - PARAMÈTRES PAR DÉFAUT

- ▶ On peut définir des **paramètres par défaut** en Python
- ▶ Ils **ne sont pas forcément** passé à l'appel de la fonction
- ▶ Dans ce cas, **la valeur spécifiée dans la définition** de la fonction sera utilisée

```
1 """ algo: passage param
2 données: nb_char -> int: taille ligne
3 résultat: affiche une ligne de taille n
4 """
5 def affiche_ligne(n: int, char: str = "*"):
6     """
7     affiche ligne avec n caractères
8     :param n: nombre de caractères
9     :return:
10    """
11    for i in range(1, n+1):
12        print(char, end="")
13
14 # données
15 nb_char: int = 10
16 affiche_ligne(nb_char, char="#")
17 print()
18 affiche_ligne(nb_char)
```

PyCharm

exercice_[1-2].py

Passage de paramètres par valeur ou par référence

PASSAGE DE PARAMÈTRES PAR VALEUR OU PAR RÉFÉRENCE

Il y a deux méthodes pour **passer des variables en paramètre** : le passage **par valeur** et le passage **par référence**

Passage par valeur

La **valeur** de l'expression passée en paramètre est **copiée dans une variable** locale : c'est cette variable qui est utilisée pour faire les calculs dans la fonction appelée

Passage par référence

La **passage par référence** consiste à passer non plus la valeur des variables comme paramètre, mais à passer les **variables elles-mêmes**

PASSAGE PAR VALEUR

- ▶ Le contenu de l'expression passée en paramètre est **copié dans la variable locale**
- ▶ **Aucune modification de la variable locale** dans la fonction appelée **ne modifie la variable passée en paramètre**, parce que ces modifications ne s'appliquent qu'à une copie de cette dernière

Python : objets **immutables** (ne peuvent pas être modifiés) sont passés par valeur

- ▶ int, float, bool, str
- ▶ tuple

EXEMPLE - PASSAGE PAR VALEUR

- ▶ La **valeur** de la variable **n** au programme principal est copiée dans le paramètre **n** de la fonction **ajoute**
 - ▶ Ils **ne portent pas** forcément le **même nom**
- ▶ **Python** : objets immutables
 - ▶ int, float, bool, str
 - ▶ tuple

```
1  """algo: passage param
2  données: n -> int
3  résultat: ajoute 10 à n
4  """
5  def ajoute(n: int):
6      """
7      ajoute 10
8      :param n: int
9      :return: n + 10
10     """
11     n = n + 10
12     print("n dans la fonction: %d" %n)
13
14 # données
15 n: int = 10
16 # opérations
17 print("n avant la fonction: %d" %n)
18 ajoute(n)
19 print("n après la fonction: %d" %n)
```


PASSAGE PAR RÉFÉRENCE

- ▶ Toute **modification du paramètre** dans la fonction appelée **entraîne la modification** de la variable passée en paramètre
- ▶ Il n'y a **plus de copie** et de variable locale avec la passage par référence

Python : objets **mutables** (peuvent être modifiés) sont passés par référence

- ▶ list
- ▶ dict
- ▶ set

EXEMPLE - PASSAGE PAR RÉFÉRENCE

- ▶ La **variable** `n` au programme principal est passé à la fonction `ajoute`
 - ▶ Ils ne **portent pas** forcément le **même nom**
- ▶ Objets mutables
 - ▶ `list`
 - ▶ `dict`
 - ▶ `set`

```
1  """algo: passage param
2  données: n -> list
3  résultat: ajoute 10 à la liste
4  """
5  def ajoute(n: list):
6      """
7      ajoute 10
8      :param n: list
9      :return:
10     """
11     n.append(10)
12     print("n dans la fonction: %s" %n)
13
14 # données
15 n: list = [10]
16 # opérations
17 print("n avant la fonction: %s" %n)
18 ajoute(n)
19 print("n après la fonction: %s" %n)
```

AVANTAGES ET INCONVÉNIENTS DES DEUX MÉTHODES

- ▶ Les **passages par valeurs** permettent d'**éviter de détruire** par mégarde les variables passées en paramètre
- ▶ Les **passages par références** sont plus **rapides** et plus **économes** en mémoire que les passages par valeur, puisque les étapes de la création de la variable locale et la copie de la valeur ne sont pas faites
- ▶ Il faut donc éviter les passages par valeur dans les **cas d'appels récursifs** de fonction ou de fonctions travaillant avec des **grandes structures de données** (matrices par exemple)
 - ▶ Les langages Python et Java gèrent la passage des paramètres

SOMMAIRE

Objective

Rappel

La passage de paramètres

Passage de paramètres par valeur ou par référence

Portée des variables

PORTÉE DES VARIABLES

EXEMPLE D'ALGORITHME

```
1. *****\n2. *****\n3. *****\n4. *****\n5. *****\n6. *****\n7. *****\n8. *****\n9. *****\n10. *****\n
```

```
1  """algo: lignes
2  données: n -> nombre de lignes
3  résultat: affiche un carré
4  """
5  def repete_caractere():
6      """
7      répète caractère * n fois
8      :return:
9      """
10     n: int = 20
11     print("*" * n)
12
13     ### programme principal
14     # données
15     n: int = 10
16     # affiche carré
17     for i in range(n):
18         repete_caractere()
```

EXEMPLE D'ALGORITHME

```
1. *****\n2. *****\n3. *****\n4. *****\n5. *****\n6. *****\n7. *****\n8. *****\n9. *****\n10. *****\n
```

```
1  """algo: lignes
2  données: n -> nombre de lignes
3  résultat: affiche un carré
4  """
5  def repete_caractere():
6      """
7      répète caractère * n fois
8      :return:
9      """
10     n: int = 20
11     print("*" * n)
12
13     ### programme principal
14     # données
15     n: int = 10
16     # affiche carré
17     for i in range(n):
18         repete_caractere()
```

EXEMPLE D'ALGORITHME

```
1. *****\n2. *****\n3. *****\n4. *****\n5. *****\n6. *****\n7. *****\n8. *****\n9. *****\n10. *****\n
```

```
1  """algo: lignes
2  données: n -> nombre de lignes
3  résultat: affiche un carré
4  """
5  def repete_caractere():
6      """
7      répète caractère * n fois
8      :return:
9      """
10     n: int = 20
11     print("*" * n)
12
13     ### programme principal
14     # données
15     n: int = 10
16     # affiche carré
17     for i in range(n):
18         repete_caractere()
```


LA PORTÉE D'UNE VARIABLE (*SCOPE*)

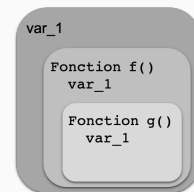
L'exemple de la procédure `repete_caractere()` met en évidence l'utilisation de **deux variables de même nom (n)** : l'une dans le programme principal lignes, l'autre dans le sous-programme `repete_caractere`

L'**endroit** où les variables **sont déclarées** est très important : selon cet endroit, les variables ont une **portée** différente

Définition

La **portée** d'une variable est **la portion de code dans laquelle elle existe**

Peut-on accéder à son contenu à cet endroit ?



LA PORTÉE D'UNE VARIABLE

- Le **cas général** dit qu'une variable n'est **visible** et **accessible** par défaut que dans le bloc d'instructions où elle a **été déclarée**

```
1  """ algo: portée
2  """
3  def g():
4      x: int = 30
5      print("g:", x)
6
7  def f():
8      x: int = 20
9      print("f:", x)
10
11  ### programme principal
12  x: int = 10
13  f()
14  g()
15  print("p:", x)
```

LA PORTÉE D'UNE VARIABLE

- Le **cas général** dit qu'une variable n'est **visible** et **accessible** par défaut que dans le bloc d'instructions où elle a **été déclarée**
- Une **variable déclarée** dans une fonction sous les mots-clés **def** ne pourra dans ce cas qu'**être lisible** et **modifiable** uniquement dans cette fonction

```
1  """ algo: portée
2  """
3  def g():
4      x: int = 30
5      print("g:", x)
6
7  def f():
8      x: int = 20
9      print("f:", x)
10
11  ### programme principal
12  x: int = 10
13  f()
14  g()
15  print("p:", x)
```

LA PORTÉE D'UNE VARIABLE

- ▶ Le **cas général** dit qu'une variable n'est **visible** et **accessible** par défaut que dans le bloc d'instructions où elle a **été déclarée**
- ▶ Une **variable déclarée** dans une fonction sous les mots-clés **def** ne pourra dans ce cas qu'**être lisible** et **modifiable** uniquement dans cette fonction
- ▶ Idem pour le **programme principal** : une variable déclarée dans cet endroit ne sera accessible que par celui-ci **par défaut**

```
1  """ algo: portée
2  """
3  def g():
4      x: int = 30
5      print("g:", x)
6
7  def f():
8      x: int = 20
9      print("f:", x)
10
11  ### programme principal
12  x: int = 10
13  f()
14  g()
15  print("p:", x)
```

LA PORTÉE D'UNE VARIABLE

```
1 def f(x: int) -> int:
2     x = x+1
3     print("dans f(x): x =", x)
4     return x
5
6 x: int = 3
7 z: int = f(x)
```

Portée globale

fCode de
f**x**

3

z

LA PORTÉE D'UNE VARIABLE

```
1 def f(x: int) -> int:
2     x = x+1
3     print("dans f(x): x =", x)
4     return x
5
6 x: int = 3
7 z: int = f(x)
```

Portée globale

f

Code de
f

x

3

z

Portée de f

x

3

LA PORTÉE D'UNE VARIABLE

```
1 def f(x: int) -> int:
2     x = x+1
3     print("dans f(x): x =", x)
4     return x
5
6 x: int = 3
7 z: int = f(x)
```

Portée globale

f

Code de
f

x

3

z

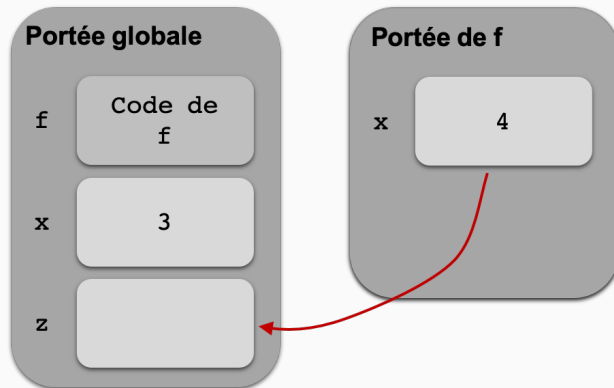
Portée de f

x

4

LA PORTÉE D'UNE VARIABLE

```
1 def f(x: int) -> int:
2     x = x+1
3     print("dans f(x): x =", x)
4     return x
5
6 x: int = 3
7 z: int = f(x)
```



LA PORTÉE D'UNE VARIABLE

```
1 def f(x: int) -> int:
2     x = x+1
3     print("dans f(x): x =", x)
4     return x
5
6 x: int = 3
7 z: int = f(x)
```

Portée globale

fCode de
f**x**

3

z

4

LA PORTÉE D'UNE VARIABLE

- ▶ à l'intérieur d'une fonction, **permet d'accéder** à une variable définie à l'extérieur
- ▶ à l'intérieur d'une fonction, **ne peut pas modifier** une variable définie à l'extérieur

```
1 def f(y: int):  
2     x: int = 1  
3     x += 1  
4     print(x)  
  
5  
6 x: int = 5  
7 f(x)  
8 print(x)
```

```
1 def g(y: int):  
2     print(x)  
3     print(x + 1)  
  
4  
5 x: int = 5  
6 g(x)  
7 print(x)
```

```
1 def h(y: int) -> int:  
2     x = x + 1  
3     print(x + 1)  
  
4  
5 x: int = 5  
6 h(x)  
7 print(x)
```

LA PORTÉE D'UNE VARIABLE

- ▶ à l'intérieur d'une fonction, **permet d'accéder** à une variable définie à l'extérieur
- ▶ à l'intérieur d'une fonction, **ne peut pas modifier** une variable définie à l'extérieur

```
1 def f(y: int):  
2     x: int = 1  
3     x += 1  
4     print(x)  
  
5  
6 x: int = 5  
7 f(x)  
8 print(x)
```

```
1 def g(y: int):  
2     print(x)  
3     print(x + 1)  
  
4  
5 x: int = 5  
6 g(x)  
7 print(x)
```

```
1 def h(y: int) -> int:  
2     x = x + 1  
3     print(x + 1)  
  
4  
5 x: int = 5  
6 h(x)  
7 print(x)
```

- ▶ différentes variables x
- ▶ ligne 2 : x est redéfini dans la portée de f

LA PORTÉE D'UNE VARIABLE

- ▶ à l'intérieur d'une fonction, **permet d'accéder** à une variable définie à l'extérieur
- ▶ à l'intérieur d'une fonction, **ne peut pas modifier** une variable définie à l'extérieur

```
1 def f(y: int):  
2     x: int = 1  
3     x += 1  
4     print(x)  
  
5  
6 x: int = 5  
7 f(x)  
8 print(x)
```

- ▶ différentes variables x
- ▶ ligne 2 : x est redéfini dans la portée de f

```
1 def g(y: int):  
2     print(x)  
3     print(x + 1)  
  
4  
5 x: int = 5  
6 g(x)  
7 print(x)
```

- ▶ x à l'intérieur de g est récupéré à partir de la portée qui a appelé g
- ▶ ligne 2 : x vient de l'extérieur de g

```
1 def h(y: int) -> int:  
2     x = x + 1  
3     print(x + 1)  
  
4  
5 x: int = 5  
6 h(x)  
7 print(x)
```

LA PORTÉE D'UNE VARIABLE

- ▶ à l'intérieur d'une fonction, **permet d'accéder** à une variable définie à l'extérieur
- ▶ à l'intérieur d'une fonction, **ne peut pas modifier** une variable définie à l'extérieur

```
1 def f(y: int):  
2     x: int = 1  
3     x += 1  
4     print(x)  
  
5  
6 x: int = 5  
7 f(x)  
8 print(x)
```

- ▶ différentes variables x
- ▶ ligne 2 : x est redéfini dans la portée de f

```
1 def g(y: int):  
2     print(x)  
3     print(x + 1)  
  
4  
5 x: int = 5  
6 g(x)  
7 print(x)
```

- ▶ x à l'intérieur de g est récupéré à partir de la portée qui a appelé g
- ▶ ligne 2 : x vient de l'extérieur de g

```
1 def h(y: int) -> int:  
2     x = x + 1  
3     print(x + 1)  
  
4  
5 x: int = 5  
6 h(x)  
7 print(x)
```

- ▶ **erreur** de portée
- ▶ ligne 2 :
UnboundLocalError:
local variable 'x'
referenced before
assignment

PyCharm

exercice_[3-4].py

CONCLUSION

Contenu vu :

- ▶ le **passage de paramètres**
- ▶ la **portée** d'une variable

RÉFÉRENCE

Algorithmique - Techniques fondamentales de programmation

Chapitre : Les sous-programmes

Ebel et Rohaut, <https://aai-logon.hes-so.ch/eni>

Cyberlearn : 19_HES-SO-GE_631-1 FONDEMENT DE LA PROGRAMMATION

(*welcome*)

<http://cyberlearn.hes-so.ch>