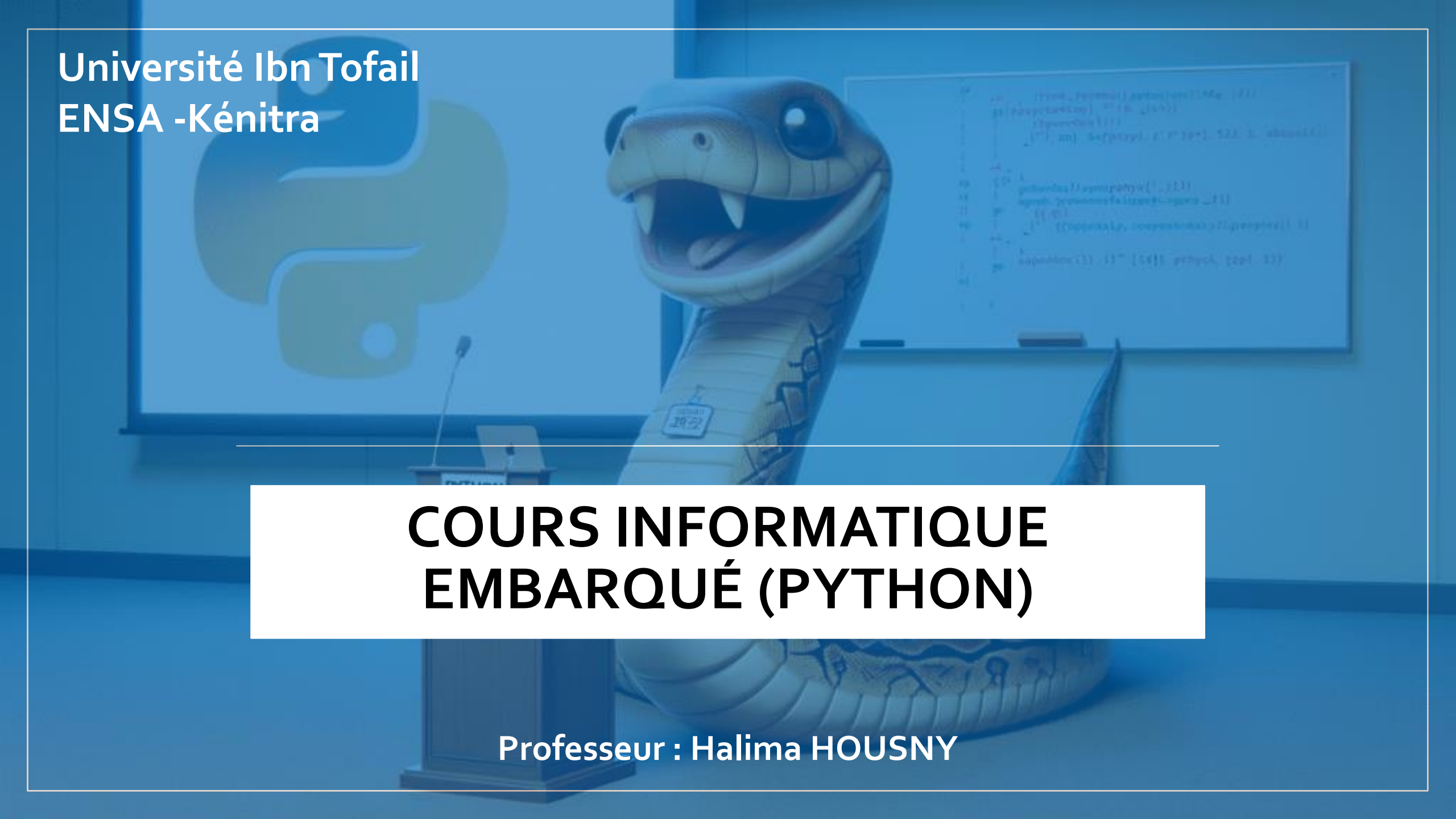
Université Ibn Tofail
ENSA -Kénitra



COURS INFORMATIQUE EMBARQUÉ (PYTHON)

Professeur : Halima HOUSNY

LES FONCTIONS

professeur : Halima HOUSNY

Les fonctions

Définition

❑ Une fonction est une suite d'instructions que l'on peut appeler avec un nom

Syntaxe :

```
def nom_fonction(liste de paramètres) :
    bloc d'instructions
```

❑ Il y a deux possibilités:

Possibilité 1: Bloc d'instructions sans retourner de valeur (Procédure)

Exemple :

```
def ma_procedure( ):
    # Instructions de la procédure
    print("Bonjour tout le monde ")
#Programme principal
#Appel de la procédure
ma_procedure()
```

```
def presentation (nom,age):
    # Instructions de la procédure
    print(f"Bonjour {nom}, tu as l'âge {age}")
#Programme principal
#Appel de la procédure
presentation("Khawla", 25)
```

Les fonctions

Définition

❑ Une fonction est une suite d'instructions que l'on peut appeler avec un nom

Syntaxe :

```
def nom_fonction(liste de paramètres) :
    bloc d'instructions
```

❑ Il y a deux possibilités:

Possibilité 2 : Bloc d'instructions avec retour de valeur (fonction)

Exemple :

```
def ma_fonction( ):
    # Instructions de la fonction
    return " Fonction Python"
#Programme principal
#Appel de la fonction
print(ma_fonction())
```

```
def cube (x):
    # Instructions de la fonction
    return x*x*x
#Programme principal
#Appel de la fonction
print(cube(5))
```

Les fonctions

L'instruction return

- L'instruction *return* signifie qu'on va renvoyer la valeur, pour pouvoir la récupérer ensuite et la stocker dans une variable par exemple.
- Cette instruction arrête le déroulement de la fonction, le code situé après le return ne s'exécutera pas.

Exemple:

```
def operations(a,b):
    somme =a+b
    soustraction=a-b
    produit =a*b
    division=a/b
    return somme ,soustraction, produit ,division
# Programme principal
sm,ss,p,d=operation (34,12)
```

l'instruction: **sm,ss,p,d=operation** (**34,12**) permet d'exécuter la fonction **operation** en remplaçant **a** par **34** et **b** par **12** et retourner les valeurs de la somme, la soustraction , le produit et la division et les stocke dans les variables sm, ss, p et d.

Les fonctions

Paramètres avec valeur par défaut

Exemples

```
def initport(speed=9600, parity="paire", data=8, stops=1):
    print("Initialisation à ", speed, "bits/s")
    print("parité :", parity)
    print(data, "bits de données")
    print(stops, "bits d'arrêt")

# Appels possibles :
initport()
```

```
Initialisation à  9600 bits/s
parité : paire
8 bits de données
1 bits d'arrêt
```

Les fonctions

Paramètres avec valeur par défaut

Exemples

```
def initport(speed=9600, parity="paire", data=8, stops=1):
    print("Initialisation à ", speed, "bits/s")
    print("parité :", parity)
    print(data, "bits de données")
    print(stops, "bits d'arrêt")

# Appels possibles :

initport(parity="nulle")
```

```
Initialisation à  9600 bits/s
parité : nulle
8 bits de données
1 bits d'arrêt
```

Les fonctions

Paramètres avec valeur par défaut

Exemples

```
def initport(speed=9600, parity="paire", data=8, stops=1):
    print("Initialisation à ", speed, "bits/s")
    print("parité :", parity)
    print(data, "bits de données")
    print(stops, "bits d'arrêt")

# Appels possibles :

initport(2400, "paire", 7, 2)
```

```
Initialisation à  2400 bits/s
parité : paire
7 bits de données
2 bits d'arrêt
```

Les fonctions

Paramètres avec valeur par défaut

Exemples

```
def initport(speed=9600, parity="paire", data=8, stops=1):
    print("Initialisation à ", speed, "bits/s")
    print("parité :", parity)
    print(data, "bits de données")
    print(stops, "bits d'arrêt")
```

```
# Appels possibles :
```

```
Initialisation à  1600 bits/s
parité : impaire
18 bits de données
10 bits d'arrêt
```

```
initport(data=18, speed=1600, stops=10, parity="impaire")
```

Les fonctions

Nombre d'arguments arbitraire : passage d'un tuple

Exemples

```
def somme(*args):
    resultat = 0
    for nombre in args:
        resultat += nombre
    return resultat
# Exemples d'appel :
print(somme(23))
print(somme(23, 42, 13))
```

```
23
78
```

Les fonctions

Nombre d'arguments arbitraire : passage d'un tuple

Exemples

```
def somme(a, b, c):
    return a + b + c
# Exemple d'appel :
elements = [2, 4, 6]
print(somme(*elements))
```

```
12
```

Les fonctions

Nombre d'arguments arbitraire : passage d'un dictionnaire

Exemples

```
def Affichage(**args):
    for k in args:
        print(f" {k} : {args[k]} ")
#Exemple d'appel
Affichage(n1=1,n2=2,n3=3)
```

```
n1 : 1
n2 : 2
n3 : 3
```

Les fonctions

Nombre d'arguments arbitraire : passage d'un dictionnaire

Exemples

```
def personne(nom, age, **kwargs):
    description = f"Nom: {nom}, Age: {age}"
    for key, value in kwargs.items():
        description += f", {key.capitalize()}: {value}"
    return description
print(personne("Jean", 30, profession="Ingénieur", ville="Paris"))
print(personne("Emilie", 25, hobby="Lecture", sport="tenis"))
```

```
Nom: Jean, Age: 30, Profession: Ingénieur, Ville: Paris
Nom: Emilie, Age: 25, Hobby: Lecture, Sport: tenis
```

Les fonctions

Les fonctions anonymes

- Python permet la création de fonctions anonymes (i.e. sans nom et donc non définie par def) à l'aide du mot-clé **lambda**.
- Une fonction anonyme ne peut pas avoir d'instruction return et doit forcément retourner une expression.
- De telles fonctions permettent de représenter tout sous forme de fonctions sans réellement en définir explicitement.

Les fonctions

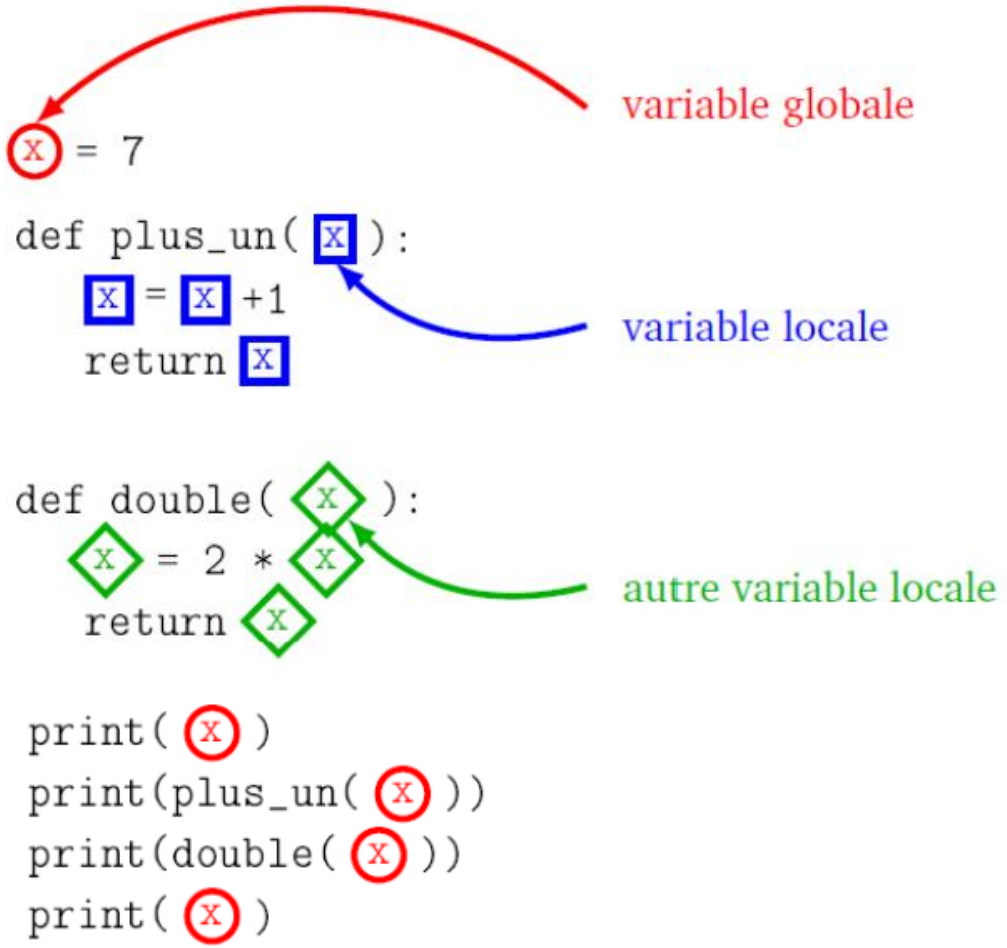
Les fonctions anonymes

Avec def	Avec lambda
def cree(n) : ** return n*n**	`cree = lambda n : n*n` `print(cree(5))`
def absolute(n) : ** if n>=0 :** ** return n** ** return -n**	`absolute = lambda n : n if n>=0 else -n` `print(absolute(-6))`
def min(a,b) : ** if a<b :** ** return a** ** return b**	`min = lambda a,b : a if a<b else b` `print(min(45,7))`

Les fonctions

Variable locale , Variable globale

- Les variables définies à l'intérieur d'une fonction sont appelées variables locales. Elles n'existent pas en dehors de la fonction.
- Une variable globale est une variable qui est définie pour l'ensemble du programme.



Les fonctions

Exercices

1. Écrire une fonction decompose(n) qui prend en entrée un entier n et retourne une liste constituée de ses chiffres. On pourra passer par des chaînes de caractères et la fonction list.
2. En déduire une fonction carres(n) qui détermine la somme des carrés des chiffres d'un entier n donné.
3. Écrire une fonction Python qui calcule et retourne la somme des nombres contenus dans un « tuple de longueur variable » .