FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Séance 6:
Fonctions & Modularité

Licence : Physique Chimie

Pr: Youssef Ouassit

Les fonctions

Plan

- Définition
- Passage d’arguments aux fonctions
- Retourner une valeur depuis une fonction
- Valeur par défaut d’un argument
- Arguments à longueur variable
- Portée des variables
- La fonction lambda

Introduction

Exemple 1: Ecrire un programme qui calcule C_n^p

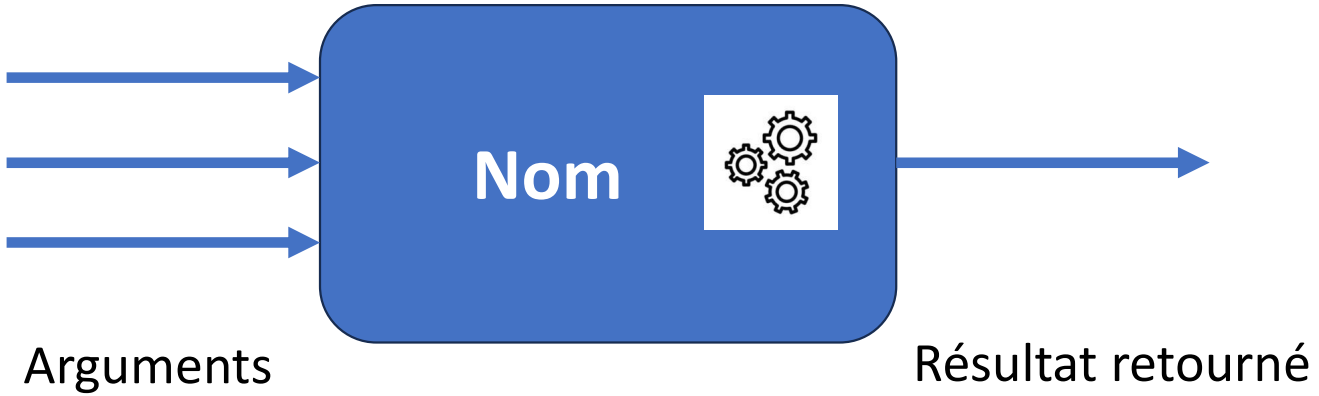
```
n = int(input("Donner n : "))
p = int(input("Donner p : "))
# calculer n!
fn = 1
for i in range(1, n+1):
    fn = fn * i
# calculer p!
fp = 1
for i in range(1, p+1):
    fp = fp * i
# calculer (n-p)!
fnp = 1
for i in range(1, n-p+1):
    fnp = fnp * i

# calculer la combinaison
C = fn // (fp*fnp)
print(C)
```

Les fonctions

Définition

Une fonction est un bloc de code réutilisable qui effectue une tâche spécifique. Elle peut prendre des paramètres (ou arguments) en entrée, effectuer des opérations sur ces données et renvoyer un résultat.

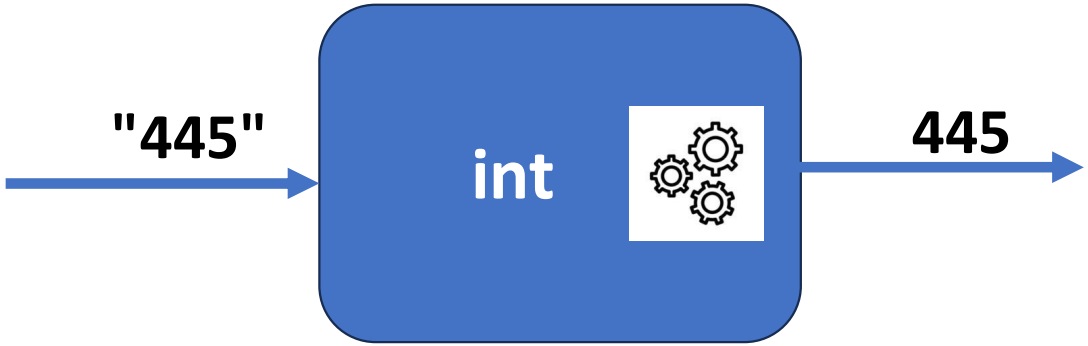


Définition

Fonctions intégrées : **sqrt**

Définition

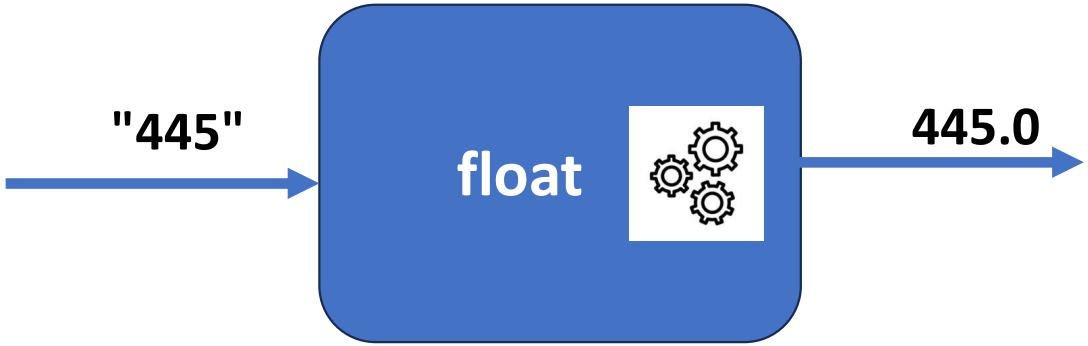
Fonctions intégrées : int



Les fonctions

Définition

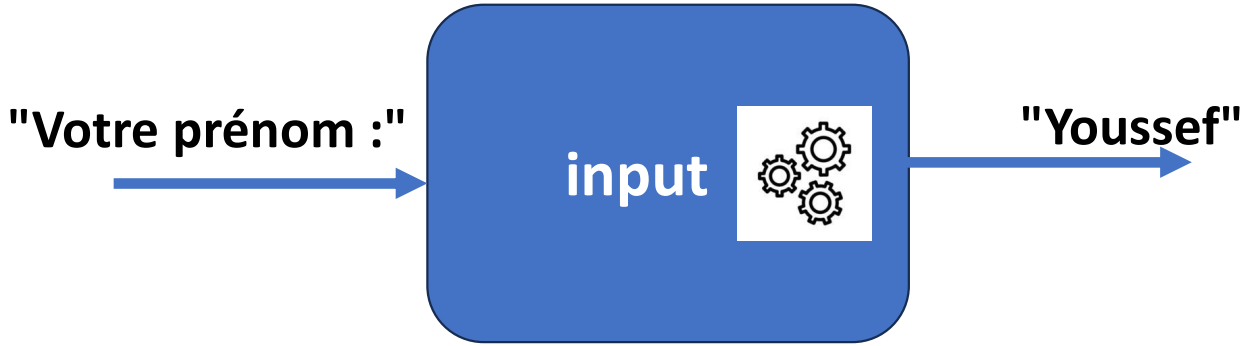
Fonctions intégrées : **float**



Les fonctions

Définition

Fonctions intégrées : **input**



Définition

Une boîte noir :

Dans le contexte de la programmation, une fonction peut être considérée comme une boîte noire si :

- **Entrées** : Tu fournis des arguments à la fonction (entrées).
- **Sorties** : La fonction renvoie un résultat (sortie).
- **Implémentation cachée** : Tu n'as pas besoin de comprendre comment la fonction réalise ses calculs internes pour l'utiliser. Tu te concentres uniquement sur ce qu'elle fait et comment l'utiliser.

Les fonctions

Syntaxe

En Python, une fonction est un bloc de code réutilisable qui effectue une tâche spécifique. Elle peut prendre des paramètres (ou arguments) en entrée, effectuer des opérations sur ces données et renvoyer un résultat. Voici comment définir une fonction :

```
def nomFonction( argument1, argument2, …) :

    instructions

    [ return résultat ]
```

Les fonctions

Syntaxe

Une fonction qui retourne le max de deux nombres :


```
def max( a, b) :
    if a>b :
        mx = a
    else :
        mx = b
    return mx
```

Les fonctions

Syntaxe

Une fonction qui retourne le max de deux nombres :


```
def max( a, b) :
    if a>b :
        return a
    else :
        return b
```

Les fonctions

Syntaxe

Une fonction qui retourne le max de deux nombres :


```
def max( a, b) :
    if a>b :
        return a
    return b
```

Les fonctions

Exemples

Une fonction qui calcule la somme de deux nombres:


```
def somme( a, b) :
    return a+b
```

Les fonctions

Exemples

Une fonction qui calcule la somme de deux nombres:


```
def fahrenheit ( c ) :
      return c*9/5  + 32
```

Les fonctions

Exemples

Une fonction qui calcule la somme de deux nombres:


```
def air_cercle ( rayon ) :
    return (22/7)*rayon**2
```

Appel d'une fonction

Une fonction peut être appelé par un programme ou une autre fonction, il suffit d'écrire le nom de la fonction et donner une valeur à chaque argument de la fonction.

Syntaxe : variable = nomFonction(Valeur1, Valeur2, …)

Exemples :

```
def max( a, b) :
    if a>b :
        return a
    return b
```

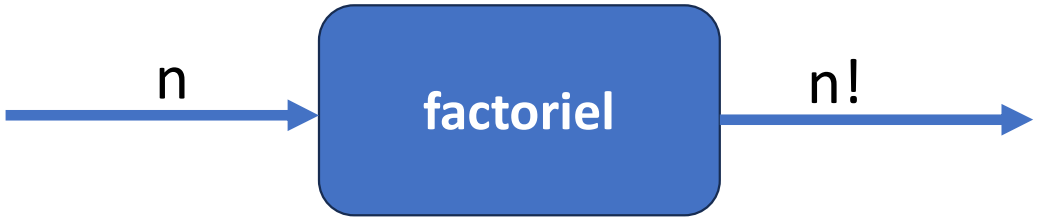
Pour appeler la fonction max :

x = max(10, 79)

y = max(39, 5)

Avantages

Exemple 1: Ecrire un programme qui calcule C_n^p




```
def factoriel(n):
    f = 1
    for i in range(2, n+1):
        f = f * i
    return f
```

Les fonctions

Avantages

Exemple 1: Ecrire un programme qui calcule C_n^p

```
n = int(input("Donner n : "))
p = int(input("Donner p : "))
# calculer n!

fn = factoriel(n)

# calculer p!

fp = factoriel(p)

# calculer (n-p)!

fnp = factoriel(n-p)


# calculer la combinaison
C = fn // (fp*fnp)
print(C)
```

Les fonctions

Avantages

Exemple 1: Ecrire un programme qui calcule C_n^p

```
n = int(input("Donner n : "))
p = int(input("Donner p : "))

fn = factoriel(n)
fp = factoriel(p)
fnp = factoriel(n-p)

# calculer la combinaison
C = fn // (fp*fnp)
print(C)
```

Les fonctions

Avantages

Exemple 1: Ecrire un programme qui calcule C_n^p

```
def factoriel(n):
    f = 1
    for i in range(1, n+1):
        f = f * i
    return f

def combinaison(n, p):
    fn = factoriel(n)
    fp = factoriel(p)
    fnp = factoriel(n-p)
    return fn // (fp*fnp)
```

```
n = int(input("Donner n : "))
p = int(input("Donner p : "))

C = combinaison(n, p)

print(C)
```

Arguments optionnels

Définition :

Python permet aux arguments de fonction d'avoir une valeur par défaut, si la fonction est appelée sans l'argument il a la valeur par défaut. De plus, les arguments peuvent être donnés dans n'importe quel ordre en utilisant les arguments nommés.

Arguments optionnels

Syntaxe :

```
def nomFonction( arg1, arg2, arg3=valeur1, arg4=valeur2) :
    instructions
    [ return résultat ]
```

Arguments optionnels

Exemple 2:

```
def montantTTC(puht , qte, tva) :
      mht = puht * qte
      mtva = mht * tva
      mttc = mht + mtva
      return mttc
```

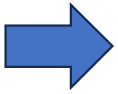
M1 = montantTTC(5000 , 10, 0.2)
M2 = montantTTC(7000 , 5 , 0.2)
M3 = montantTTC(3000 , 10, 0.2)
M4 = montantTTC(100 , 10 , 0.07)

Arguments optionnels

Exemple 2:

```
def montantTTC(puht , qte, tva=0.2) :
    mht = puht * qte
    mtva = mht * tva
    mttc = mht + mtva
    return mttc
```

M1 = montantTTC(5000 , 10, 0.2)
M2 = montantTTC(7000 , 5 , 0.2)
M3 = montantTTC(3000 , 10, 0.2)
M4 = montantTTC(100 , 10 , 0.07)



M1 = montantTTC(5000 , 10)
M2 = montantTTC(7000 , 5)
M3 = montantTTC(3000 , 10)
M4 = montantTTC(100 , 10 , 0.07)

Arguments optionnels

Exemple 2:

```
def montantTTC(puht , qte=10, tva=0.2) :
    mht = puht * qte
    mtva = mht * tva
    mttc = mht + mtva
    return mttc
```

M1 = montantTTC(5000)
M2 = montantTTC(7000 , 5)
~~**M4 = montantTTC(100 , 0.07)**~~
M4 = montantTTC(100 , tva = 0.07)

Les fonctions lambda

Les fonctions lambda en Python sont des fonctions anonymes, c'est-à-dire des fonctions sans nom, qui peuvent avoir un nombre quelconque d'arguments mais ne peuvent contenir qu'une seule expression. Elles sont souvent utilisées pour des opérations simples et courtes.

La syntaxe : lambda arguments : expression

Les fonctions lambda

Exemples:

```
def f(x):
    return x**2
```

Equivalent à :

```
f = lambda x : x**2
```

```
def g(x, y):
    return x+y
```

Equivalent à :

```
g = lambda x,y : x+y
```

Porté de variables

La « portée » d'une variable désigne l'espace du script dans laquelle elle va être accessible et visible.

On distingue deux types de portée de variables :

1. **Variable locale**
2. **Variable globale**

Porté de variables

1. Variable locale:

Est une variable déclarée à l'intérieur d'une fonction. Elle n'est accessible qu'à l'intérieur de la fonction dans laquelle elle a été déclaré.

```
def ma_fonction():
        x = 4
        print('ma variable vaut ', x)

ma_fonction()

print(x)
```

Erreur : x est non définie

2. Variable globale:

Est une variable déclarée en dehors des fonctions. Elle visible et utilisable dans toute les fonctions (**mais la fonction ne peut le modifier**).

```
total = 10
def ma_fonction():
    x = total + 4
    print('ma variable vaut ', x)

ma_fonction()
print(total)
```