PYTHON

Présenté par :

Xavier TABUTEAU

Les fonctions prédéfinies

- **Quelques fonctions prédéfinies / natives (builtin)**

```
abs(x)          # retourne la valeur absolue de x.
all(iterable)   # retourne True si toutes les valeurs de l'itérable sont True.
any(iterable)   # retourne True si au moins une valeur de l'itérable est True.
bin(int)        # convertit nombre entier en chaîne de caractère binaire.
hex(int)        # convertit nombre en valeur hexadécimale.
len(obj)        # retourne la longueur de l'objet.
list(obj)       # cast l'objet au format liste.
map(fct, obj)   # applique une transformation à tous les éléments d'un itérable.
filter(fct, list)   # filtre avec un prédicat les éléments d'un itérable et retourne un booléen.
```

- **Fonctions natives de l'objet str**

```
str.capitalize()    # retourne la string avec une majuscule en début de phrase
str.title()         # retourne la string avec une majuscule en début de chaque mot
str.upper()         # retourne la string en majuscule
str.lower()         # retourne la string en minuscule
str.strip()         # retourne la string str en supprimant les espaces avant et après la phrase
str.count(x)        # retourne le nombre d'occurrence de x dans str
str.endswith("x")   # retourne True si la str fini par 'x'
str.startswith("x") # retourne True si la str commence par 'x'
str.find(x)         # retourne l'index de la première occurrence de x (-1 si n'existe pas)
```

ex_chaines

Print et Input

- **Fonction print()**

Elle permet d'afficher sur la sortie standard (paramètre file="sys.stdout"), n'importe quel nombre de valeurs fournies en arguments (c'est-à- dire entre les parenthèses). Par défaut, ces valeurs seront séparées les unes des autres par un espace (paramètre sep=" "), et le tout se terminera par un saut à la ligne (paramètre end="\n"). Le paramètre flush=False permet de ne pas afficher immédiatement la sortie, cela passe par un tampon gérer par Python.

```
>>> print("Bonjour", "à", "tous", sep ="*")      # Bonjour*à*tous sur la sortie standard
>>> print("Bonjour", "à", "tous", sep ="" , file=sys.stderr)          # Bonjouràtous sur la sortie d'erreur
```

- **Fonction input() : Interaction avec l'utilisateur**

Cette fonction provoque une interruption dans le programme courant. 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.

```
prenom = input("Entrez votre prénom : ")
print("Bonjour,", prenom)
OU
print("Veuillez entrer un nombre positif quelconque : ")
ch = input()
nn = int(ch) # conversion de la chaîne en un nombre entier
print("Le carré de", nn, "vaut", nn ** 2)
```

ex_fonctions_natives
ex_input
ex_boucles_imbriquees

Les fonctions

L’approche efficace d’un problème complexe consiste souvent à le décomposer en plusieurs sous-problèmes plus simples. D’autre part, il arrivera souvent qu’une même séquence d’instructions doit être utilisée à plusieurs reprises dans un programme. La déclaration d’une fonction commence par le mot clé « def » suivi de son nom, puis, des parenthèses pouvant contenir des arguments, et finie par « : ». Cela constitue sa signature. Son nom est aussi appelé identificateur. Son contenu c’est son implémentation. Si on ne veut pas l’implémenter tout de suite on doit écrire le mot clé « pass ».

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

- **Les arguments des fonctions**

Une fonction peut avoir un nombre fixe ou variable d’arguments. Ces arguments peuvent être des variables (int, string, list…) mais aussi d’autres fonctions. Les arguments peuvent être positionnels ou nommés. Dans certains cas, vous pouvez obliger les appelants de votre fonction particulière à spécifier les arguments en utilisant uniquement leurs noms afin d'améliorer la lisibilité du code ou d’attirer l’attention sur l’importance du paramètre.

Pour obtenir un nombre variable de paramètre à transmettre à une fonction, on préfixe le ou les arguments de la fonction par un ou deux astérisques.
Un * correspond aux arguments positionnels (sous forme de tuple).
Deux ** correspond aux arguments nommés (sous forme de dictionnaire).
Ils sont souvent nommés *args et **kwargs.

Les fonctions

- **Variable locale vs variables globales**

Lorsque nous définissons des variables à l'intérieur du corps d'une fonction, ces variables ne sont accessibles qu'à la fonction elle-même. On dit que ces variables sont des variables locales à la fonction.
Les variables définies à l'extérieur d'une fonction sont des variables globales. Leur contenu est « visible » de l'intérieur d'une fonction, mais la fonction ne peut pas le modifier.

Les fonctions

- **Fonction récursive**

Une fonction récursive est une fonction qui s'appelle elle-même. Attention à bien mettre une condition pour sortir de l'appel récursif, sinon on provoque une boucle infinie. Dans certains cas cela peut être intéressant, mais la plupart du temps ce n'est pas une bonne idée car elles sont peu performantes et il y a une limite de nombre d'appels (999 empilables en comptant la fonction main() et le premier appel de la fonction récursive.

Exemple la fonction factorielle(x)

```
def factorielle(n):
        If n < 2:
            return 1
        else:
       return n *  factorielle(n-1)
```

```
def factorielle(n):
        return 1 if n < 2 else n * factorielle(n-1)
```

fonctions.py
ex_fonction
ex_fonctions

Les fonctions

- **Fonction Lambda**

Pour Python, c'est la seule façon d'écrire une fonction anonyme. Ceci est particulièrement utile pour la programmation fonctionnelle. En effet, une fonction peut être directement écrite dans un appel de fonction sans avoir à la définir au préalable.

```
>>> list(map(lambda x: x ** 2, range(10)))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

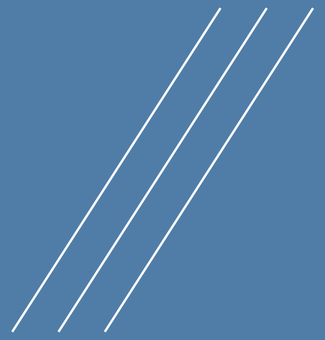
Bien quel les fonctions lambda soient utilisées dans le but de créer des fonctions anonymes, on peut décider tout de même de leur donner un nom :

```
>>> f = lambda x: x ** 2
>>> f(5)
25
>>> list(map(f, range(10)))
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Syntaxe pour passer une variable à l'exécution : lambda x: monCalculAvecX

Syntaxe pour passer une variable à la création : lambda x = x: monCalculAvecX

PYTHON



Présenté par

Xavier TABUTEAU