new idea

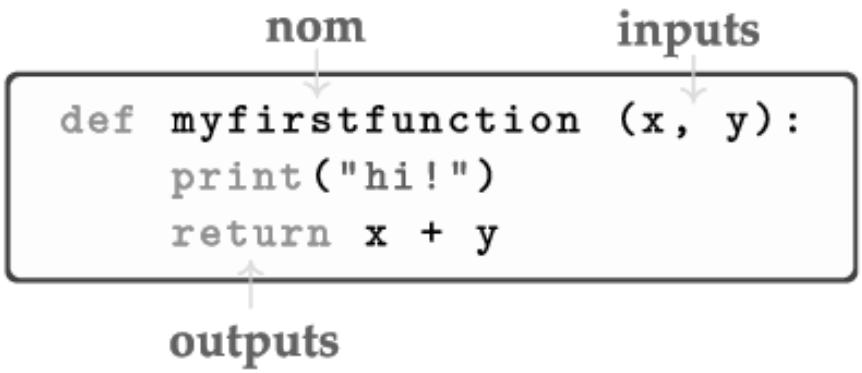
FONCTIONS

Functions en Python:

```
A = [3, 6, 12, 312, 9, -3]
c = len(A)
print(c)
```

```
6
```

Pour une fonction:



Pour utiliser une fonction:

```
a = 10
b = 20
c = myfirstfunction(a,b)
print(c)
```

```
def myfirstfunction (x, y):
    print("hi!")
    return x + y

a = 10
b = 20
c = myfirstfunction(a,b)
print(c)
```

c	30
a	10
b	20

```
hi!
```

```
30
```

```
def myfirstfunction (x, y)

    myfirstfunction(a, b)
```

les deux sont équivalents !

```
def myfirstfunction (x, y):
    print("hi!")
    return x + y

a = 10
b = 20
c = myfirstfunction(a,b)
print(c)
```

```
a = 10
b = 20

x = a
y = b
print("hi!")
c = x + y

print(c)
```

```
hi!
30
```

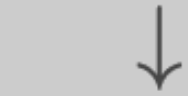
new idea

FONCTIONS

déjà existant

Some examples of functions already present in Python:

```
print( max(-1, -5, -3, 3.4, 99, 99.98, 99.98001) )
```



```
99.98001
```

```
print( min(-1, -5, -3, 3.4) )
```

```
-5
```

Some examples of functions already present in Python:

```
a = 9.43341
b = int(a)
print(b)
```

```
9
```

```
print( abs(-1.44) )
```

```
1.44
```

En Python, les `modules` contiennent des fonctions

```
a = 9.454
b = sqrt(a)
print(b)
```

→

```
Traceback (most recent call last):
  File "/home/nabil/tmp/a.py",
line 2, in <module>
    b = sqrt(a)
NameError: name 'sqrt' is not
defined
```

```
import math

a = 9.454
b = math.sqrt(a)
print(b)
```

→

```
3.0747357610045127
```

new idea

FONCTIONS

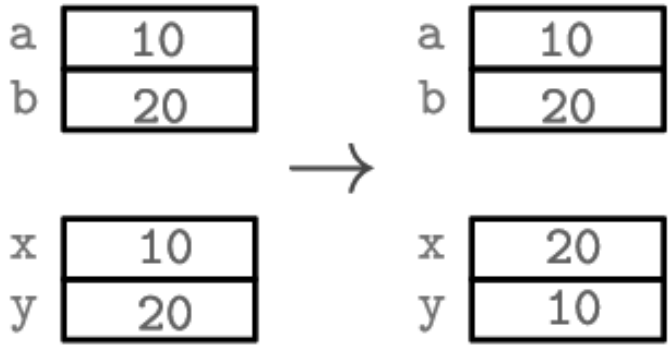
passing arguments

```
def echange(x, y):
    tmp = x
    x = y
    y = tmp

a = 10
b = 20
print("Before:", a, b)
echange(a, b)
print("After:", a, b)
```

fails!

```
Before: 10 20
After: 10 20
```

```
def echange(x, y):
    tmp = x
    x = y
    y = tmp

a = 10
b = 20
print("Before:", a, b)
echange(a, b)
print("After:", a, b)
```



les deux sont équivalents !

```
Before: 10 20
After: 10 20
```

```
a = 10
b = 20
print("Before:", a, b)

x = a
y = b
tmp = x
x = y
y = tmp

print("After:", a, b)
```

Pour `int`, `string`, `float`, `bool` : la valeur est copiée

```
def myfunction(x):
    x = 20

a = 10
print("Before:", a)
myfunction(a)
print("After:", a)
```

↓

```
Before: 10
After: 10
```

```
def myfunction(x):
    x = "again"

a = "hello"
print("Before:", a)
myfunction(a)
print("After:", a)
```

↓

```
Before: hello
After: hello
```

```
def myfunction(x):
    x = False

a = True
print("Before:", a)
myfunction(a)
print("After:", a)
```

↓

```
Before: True
After: True
```

Raison : l'affectation '=' crée une nouvelle variable

```
a = 10
print("Before: ", a)

x = a
x = 20

print("After:", a)
```

Pour une **liste** : l'affectation '=' donne un nouveau nom à la **même variable**

```
def mysecondfunction(B):
    tmp = B[0]
    B[0] = B[1]
    B[1] = tmp

A = [10, 20]
print("Before:", A[0], A[1])
mysecondfunction(A)
print("After:", A[0], A[1])
```

```
A = [10, 20]
print("Before:", A[0], A[1])

B = A
tmp = B[0]
B[0] = B[1]
B[1] = tmp

print("After:", A[0], A[1])
```

↓ ↓

```
Before: 10 20
After: 20 10
```

```
A = [3, 23, 22, 12, -3, 100, 80]

max = A[0]
for i in range(0, len(A)):
    if A[i] > max:
        max = A[i]

print("Maximum of A is:", max)
```

→

```
Maximum of A is: 100
```

créer des fonctions
pour des tâches spécifiques

```
def computeMaximum(A):
    max = A[0]

    for i in range(0, len(A)):
        if A[i] > max:
            max = A[i]

    return max

print("Maximum of A is:", computeMaximum( [3, 23, 22, 12, -3, 100, 80] )
```

↓

```
Maximum of A is: 100
```

```
def computeMaximum(A):
    max_index = 0

    for i in range(0, len(A)):
        if A[i] > A[max_index]:
            max_index = i

    print("Maximum of A is:", A[max_index], "at index:", max_index)

computeMaximum( [3, 23, 22, 12, -3, 100, 80] )
```

↓

```
Maximum of A is: 100 at index: 5
```

pas possible d'appeller une fonction qui n'est pas encore définie au moment de l'appel

```
a = 10
b = 20
c = myfirstfunction(a,b)
print(c)

def myfirstfunction (x, y):
    print("hi!")
    return x + y
```

```
Traceback (most recent call last):
  File "/home/nabil/tmp/t.py", line 3, in <module>
    c = myfirstfunction(a,b)
NameError: name 'myfirstfunction' is not defined
```

Une fonction peut renvoyer plusieurs variables

```
def mysecondfunction(x):
    return 2*x, 3*x, 4*x

a,b,c = mysecondfunction(5)
print(a,b,c)
```

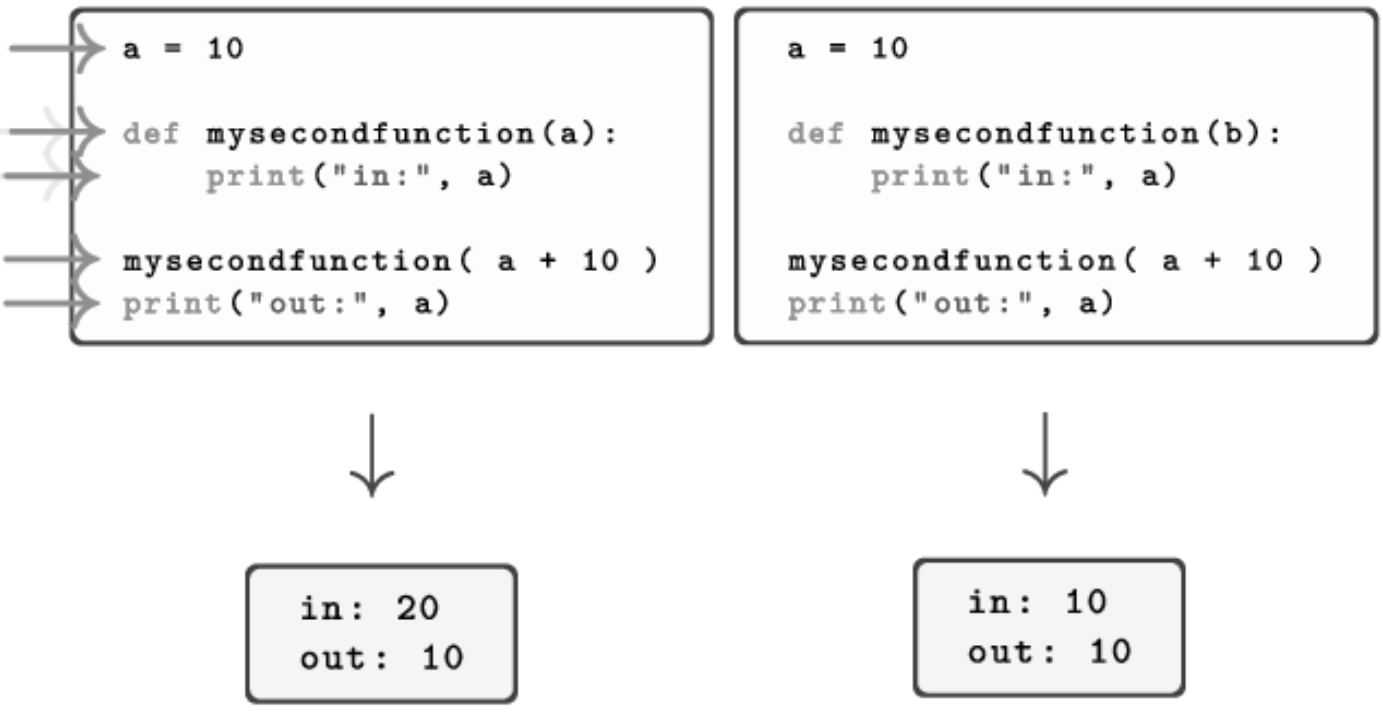
```
10 15 20
```

new idea

VARIABLES:

Global

Les noms des variables locales sont prioritaires



La variable a est une nouvelle variable locale

```
a = 10

def f():
    a = 20

print("a1: ", a)
f()
print("a2: ", a)
```

```
a1: 10
a2: 10
```

Utiliser le mot-clé `global` pour accéder à la variable globale

```
a = 10

def f():
    global a
    a = 20

print("a1:", a)
f()
print("a2:", a)
```

```
a1: 10
a2: 20
```

new idea

FONCTIONS

nommer arguments

defaut arguments

les derniers arguments peuvent avoir des valeurs par défaut

```
def f(x,y='hi'):
    print("x: ", x, " and y: ", y)

f(10,4)
f(10)
```

```
x:  10  and y:  4
x:  10  and y:  hi
```

```
def f(x = 'hello',y):
    print("x: ", x, " and y: ", y)

f(10,4)
```

les arguments avec valeurs par défaut doivent être placés à la fin

```
  File "/home/nabil/tmp/t.py", line 1
    def f(x = 'hello',y):
                      ^
SyntaxError: parameter without a default follows parameter with a default
```

ordre des arguments lors de la définition d'une fonction = ordre des arguments lors de l'appel d'une fonction

```
def f(x,y):
    print( x**2 )
    print("My name is: ", y)

f(5, "john")
```

→

```
25
My name is:  john
```

```
def f(x,y):
    print( x**2 )
    print("My name is: ", y)

f("john", 5)
```

→

```
Traceback (most recent call last):
  File "/home/nabil/tmp/t.py",
       line 6, in <module>
    f("john", 5)
  File "/home/nabil/tmp/t.py", line 2, in f
    print( x**2 )
TypeError: unsupported operand type(s)
for ** or pow(): 'str' and 'int'
```

Vous **pouvez** modifier l'ordre lors de l'appel, mais en utilisant les **noms** des variables

```
def f(x,y):
    print( x**2 )
    print("My name is: ", y)

f(y="john", x=5)
```

→

```
25
My name is:  john
```

Utiliser un * pour **obliger** de nommer chaque argument

```
def f(*, x,y):
  print("x: ", x, " and y: ", y)

f(x=10,y=4)
```

```
x:  10  and y:  4
```

```
def f(*, x='hello',y):
  print("x: ", x, " and y: ", y)

f(x=10,y=4)
f(y=2)
f(y=5, x=True)
```

```
x:  10  and y:  4
x:  hello  and y:  2
x:  True  and y:  5
```

```
def f(*, x,y):
  print("x: ", x, " and y: ", y)

f(10, 4)
```

```
Traceback (most recent call last):
  File "/home/nabil/tmp/t.py", line 4, in <module>
    f(10,4)
TypeError: f() takes 0 positional arguments but 2 were given
```

new idea

FONCTIONS

fonctions comme argument

Nous pouvons même envoyer des fonctions comme arguments

```
def f(g):
    print( type(g) )
    g("hi!")


f( print )
```

```
<class 'builtin_function_or_method'>
hi!
```

```
def k(x,y):
    print(x, "+", y, "is:", x+y)

def f(g):
    print( type(g) )
    g(4,10)


f(k)
```

```
<class 'function'>
4 + 10 is: 14
```

new idea

FONCTIONS: CATÉGORIES

1. Fonctions que vous écrivez vous-même

```
def mysecondfunction(x):
    return 2*x, 3*x, 4*x

a,b,c = mysecondfunction(5)
print(a,b,c)
```

→

```
10 15 20
```

2. Fonctions qui sont disponibles globalement

```
A = [3, 7, 1, 6, 788]

n = len(A)

print(n)
```

→

```
5
```

3. Fonctions qui appartiennent à des types de variables

```
A = [3, 7, 1, 6, 788]
A.append("hello")
print(A)
```

→

```
[3, 7, 1, 6, 788, 'hello']
```

```
n = 10
n.append(20)
```

→

```
Traceback (most recent call last):
  File "/home/nabil/tmp/t.py", line 2, in <module>
    n.append(20)
    ^^^^^^^^^
AttributeError: 'int' has no attribute 'append'
```