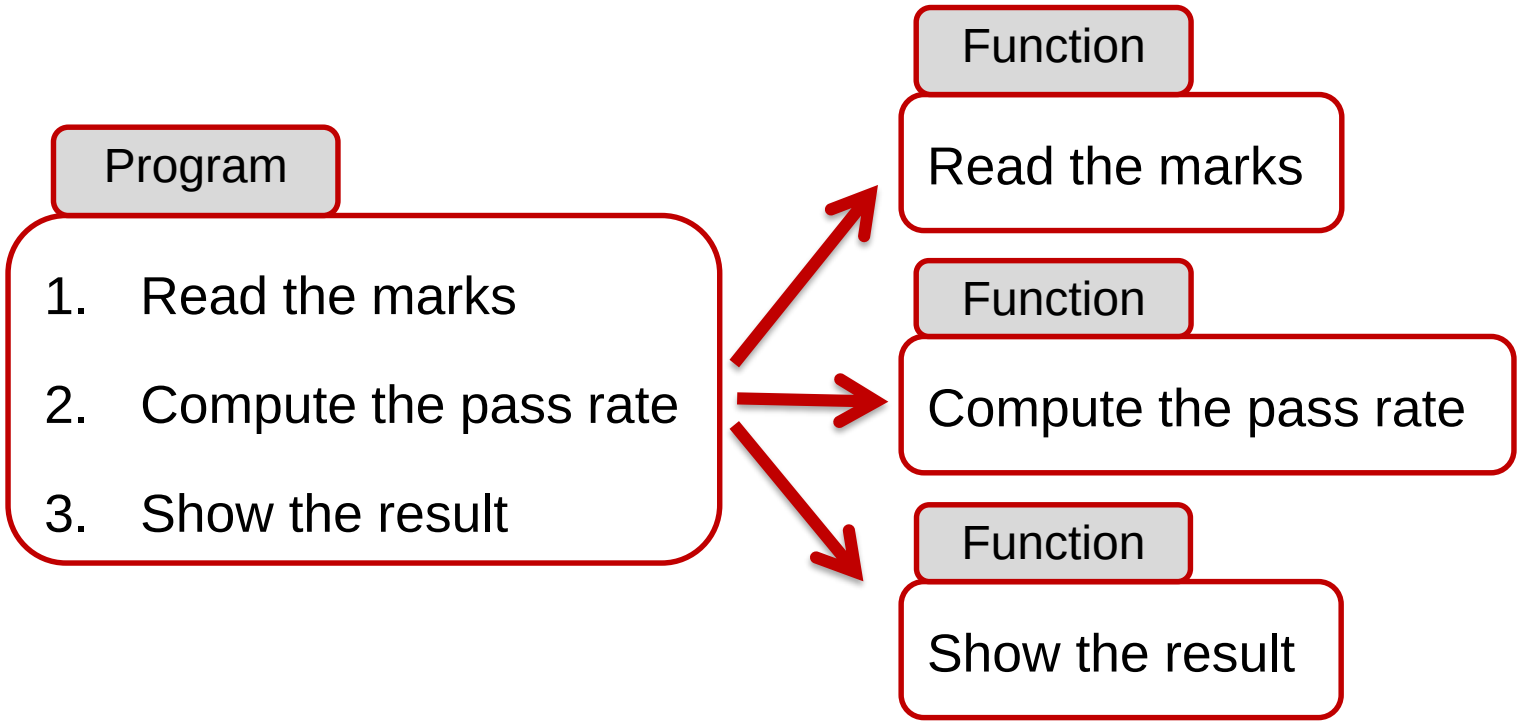
Table of contents

Functions

- A function is a fragment of code in a program that solves a sub-problem with its own entity
- **Example:** Write a program to read the marks of the students, compute the number of pass rates and show it on the screen



Definition and use

Reserved word

Function name

```
def NAME(PARAMETER LIST):
    SENTENCES
```

Data that the function needs

Function code

Definition and use

```
def new_line():
    print()

print("First line.")
new_line()
print("Second line.")
```

Output

```
First line.

Second line.
```

Definition and use

```
def new_line():
    print()

print("First line.")
new_line()
new_line()
new_line()
print("Second line.")
```

```
def new_line():
    print()

def three_lines():
    new_line()
    new_line()
    new_line()

print("First line.")
three_lines()
print("Second line.")
```

Output

```
First line.


Second line.
```

Advantages of using functions

- They **group sentences** and they give them a name.
- **Simplified code:** behind a function call can be a complex code
- **Shorter programs:** redundant code no longer exists
- They allow the **sharing** of the workload among programmers

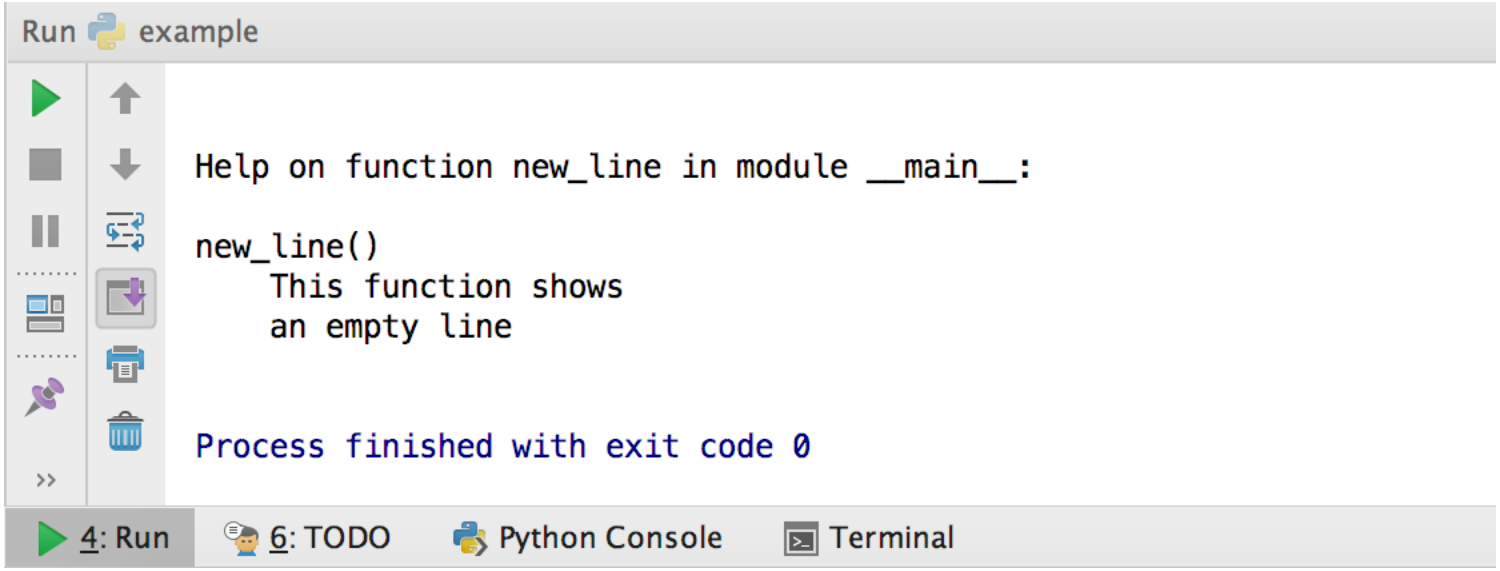
Function documentation

```
def new_line():
    """This function shows
    an empty line"""
    print()    # it prints an empty line

def three_lines():
    """This function shows
    three empty lines"""
    new_line()    # it calls 3 times the function new_line
    new_line()
    new_line()

print("First line.")
three_lines()
print("Second line.")
```

Use three double quotes (""") at the beginning of the function

Function documentation

```
help(new_line)
```



Parameters and arguments

- What if we want to draw a line of 10 asterisks (*) and 20 hyphens (-)?
- Is it necessary to implement two different functions?

Solution: Implement one function with parameters

Parameters and arguments

```
def draw_line(times, char):
    """"Shows a line on the screen with a given number
    of characters. It has 2 parameters:
    - times: number of times the character will appear
    - char: character to be shown"""

    i = 1
    string = ""
    while i <= times:
        string = string + char
        i = i + 1
    print(string)
```

Parameters and arguments

Definition

```
def draw_line(times, char):
    i = 1
    string = ""
    while i <= times:
        string = string + char
        i = i + 1
    print(string)
```

Function call

```
draw_line(10, "*")

draw_line(20, "-")
```

Execution flow

Definition

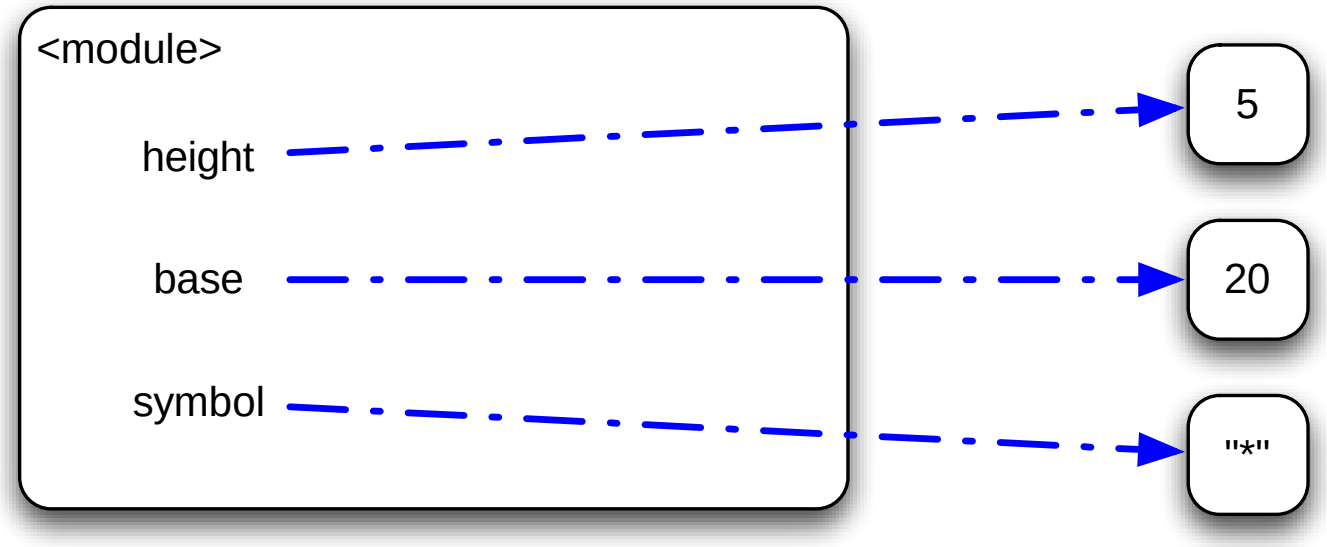
```
def draw_line(times, char):
    i = 1
    string = ""
    while i <= times:
        string = string + char
        i = i + 1
    print(string)

def draw_rectangle(height, width, char):
    i = 1
    while i <= height:
        draw_line(width, char)
        i = i + 1
```

Program starts here

```
height = 5
base = 20
symbol = "*"
draw_rectangle(height, base, symbol)
```

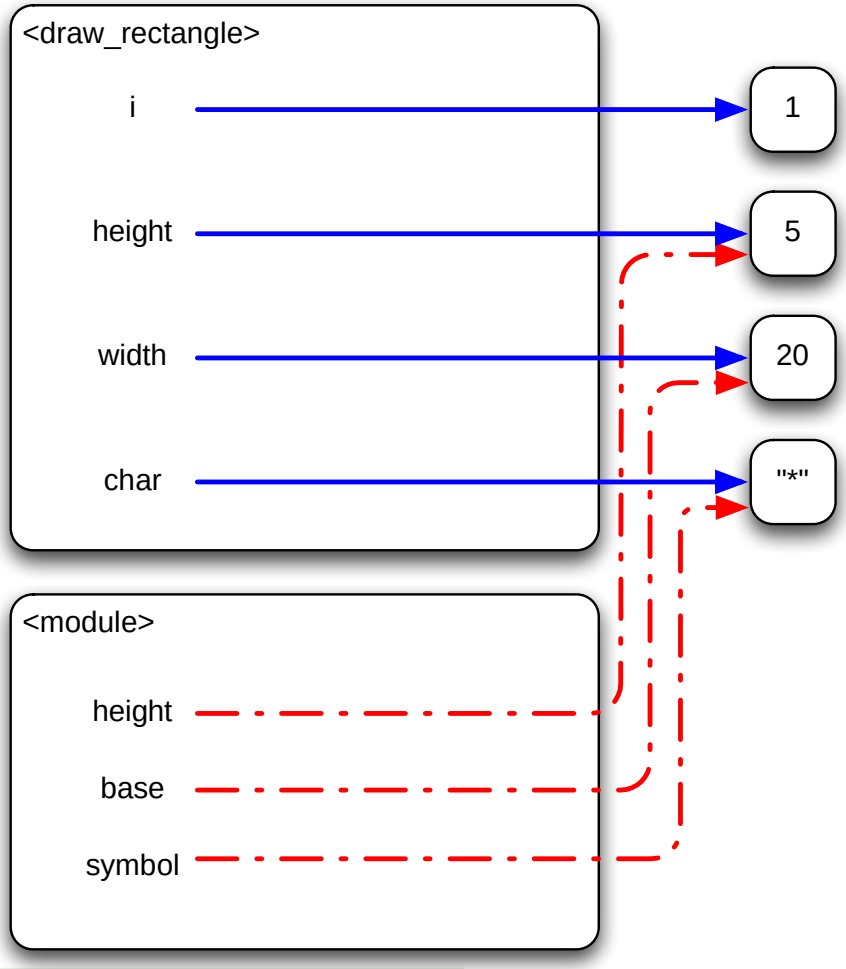
Execution flow

```
height = 5
base = 20
symbol = "*"
draw_rectangle(height, base, symbol)
```



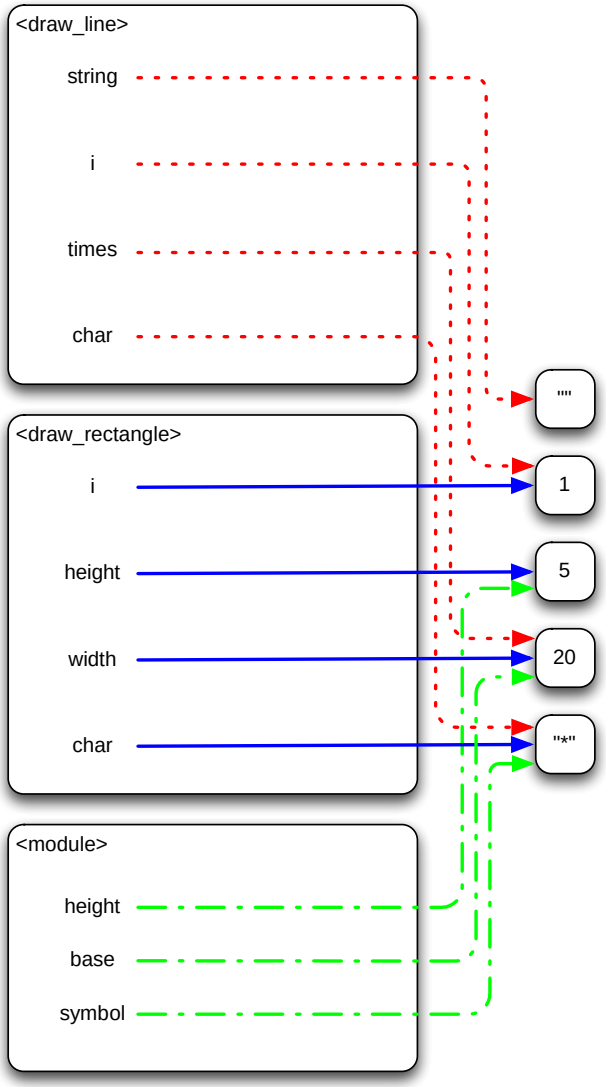
Execution flow

```
def draw_rectangle(height, width, char):
    i = 1
    while i <= height:
        draw_line(width, char)
        i = i + 1
```



Execution flow

```
def draw_line(times, char):
    i = 1
    string = ""
    while i <= times:
        string = string + char
        i = i + 1
    print(string)
```



Error messages

```
def draw_line(times, char):
    i = 1
    string = ""
    while i <= times:
        string = string + char
        i = i + char    # error on purpose
    print(string)

def draw_rectangle(height, width, char):
    i = 1
    while i <= height:
        draw_line(width, char)
        i = i + 1

height = 5
base = 20
symbol = "*"
draw_rectangle(height, base, symbol)
```

Error messages

```
def draw_line(times, char):
    i = 1
    string = ""
    while i <= times:
        string = string + char
        i = i + char    # error on purpose
    print(string)

def draw_rectangle(height, width, char):
    i = 1
    while i <= height:
        draw_line(width, char)
        i = i + 1

height = 5
base = 20
symbol = "*"
draw_rectangle(height, base, symbol)
```

```
Traceback (most recent call last):
  File "example.py", line 18, in <module>
    draw_rectangle(height,base,symbol)
  File "example.py", line 12, in draw_rectangle
    pinta_linea(ancho,car)
  File "example.py", line 6, in draw_line
    i = i + char
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Modifying the value of the parameters

- What happens when the value of a parameter is modified **inside** a function?

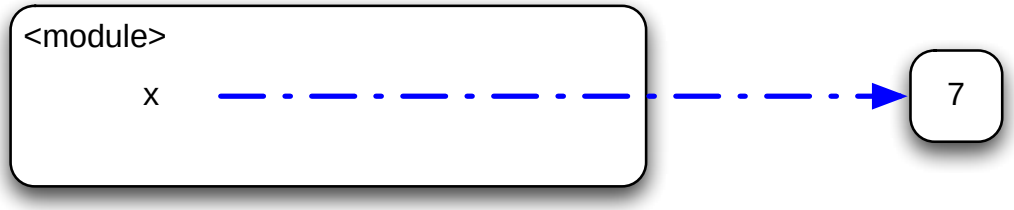
```
def decrement(val):
    val = val - 1
    print (val)

x = 7
print(x)
decrement(x)
print(x)
```

- What values will appear on the screen?

Modifying the value of the parameters

Output

```
def decrement(val):
    val = val - 1
    print(val)

x = 7
print(x)
decrement(x)
print(x)
```

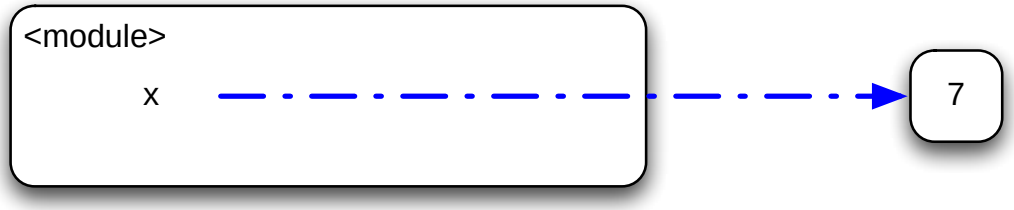


Modifying the value of the parameters

Output

7

```
def decrement(val):
    val = val - 1
    print(val)

x = 7
print(x)
decrement(x)
print(x)
```

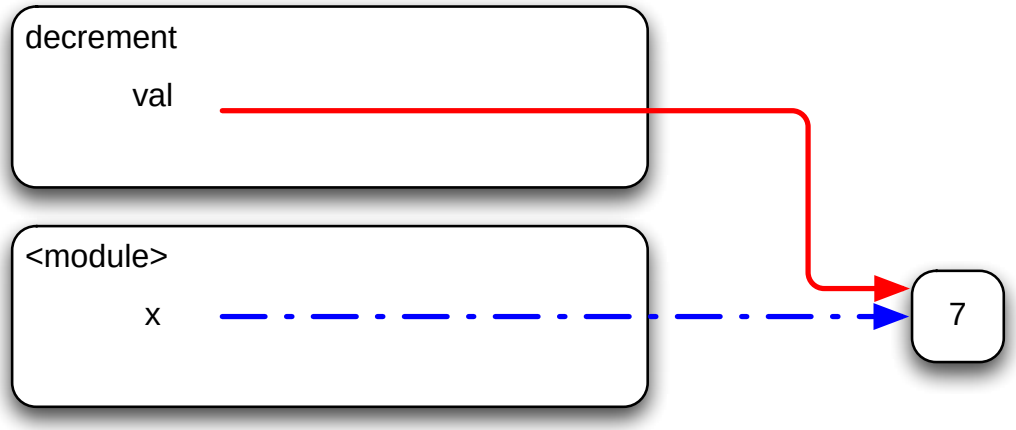


Modifying the value of the parameters

Output

7

```
def decrement(val):
    val = val - 1
    print(val)

x = 7
print(x)
decrement(x)
print(x)
```



Modifying the value of the parameters

Output

7

```
def decrement(val):
    val = val - 1
    print(val)

x = 7
print(x)
decrement(x)
print(x)
```

Modifying the value of the parameters

Output

```
7
6
```

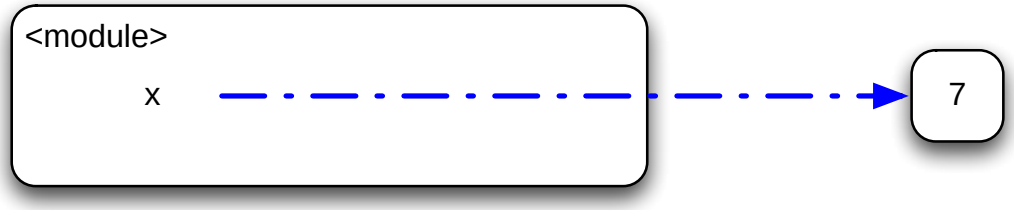
```
def decrement(val):
    val = val - 1
    print(val)

x = 7
print(x)
decrement(x)
print(x)
```

Modifying the value of the parameters

Output

```
7
6
7
```

```
def decrement(val):
    val = val - 1
    print(val)

x = 7
print(x)
decrement(x)
print(x)
```



Variable and parameter scope

- **Variables and parameters** declared **inside** a function are considered **local**, as they can only be accessed from inside the same function.
- Variables declared **outside** functions are considered **global** and can be accessed from **anywhere** in the program. In order to access a global variable within a function, we have to declare it using the reserved word `global`.
  - **Example:** `global a`

Variable and parameter scope

```
def draw_line(times, char):
    i=1
    string=""
    while i <= times:
        string = string + char
        i = i + 1
    print(string)

def draw_rectangle(height, width, char):
    i=1
    while i <= height:
        draw_line(width,char)
        i = i + 1

height = 5
base = 20
symbol = "*"
draw_rectangle(height, base, symbol)
```

Variable and parameter scope

```
def draw_line(times, char):
    i=1
    string=""
    while i <= times:
        string = string + char
        i = i + 1
    print(string)

def draw_rectangle(height, width, char):
    i=1
    while i <= height:
        draw_line(width,char)
        i = i + 1

height = 5
base = 20
symbol = "*"
draw_rectangle(height,base,symbol)
```

Variable and parameter scope

```
def draw_line(times, char):
    i=1
    string=""
    while i <= times:
        string = string + char
        i = i + 1
    print(string)

def draw_rectangle(height, width, char):
    i=1
    while i <= height:
        draw_line(width,char)
        i = i + 1

height = 5
base = 20
symbol = "*"
draw_rectangle(height, base, symbol)
```

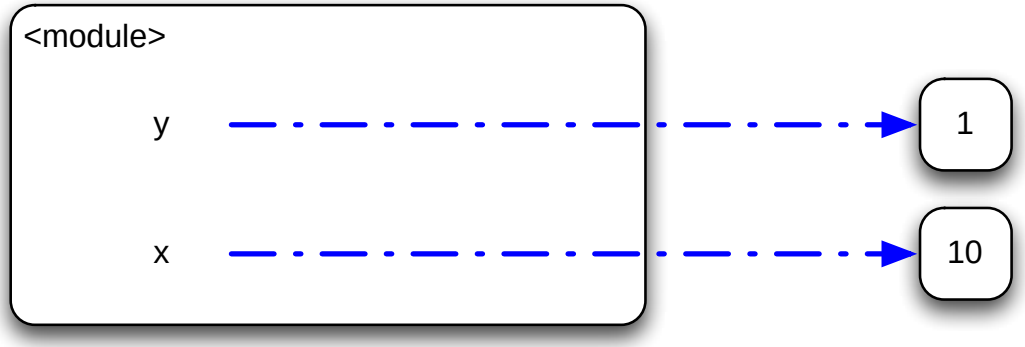
Parameter and variable scope

- What happens when several variables share the same name in different scopes?

The local variable is accessed

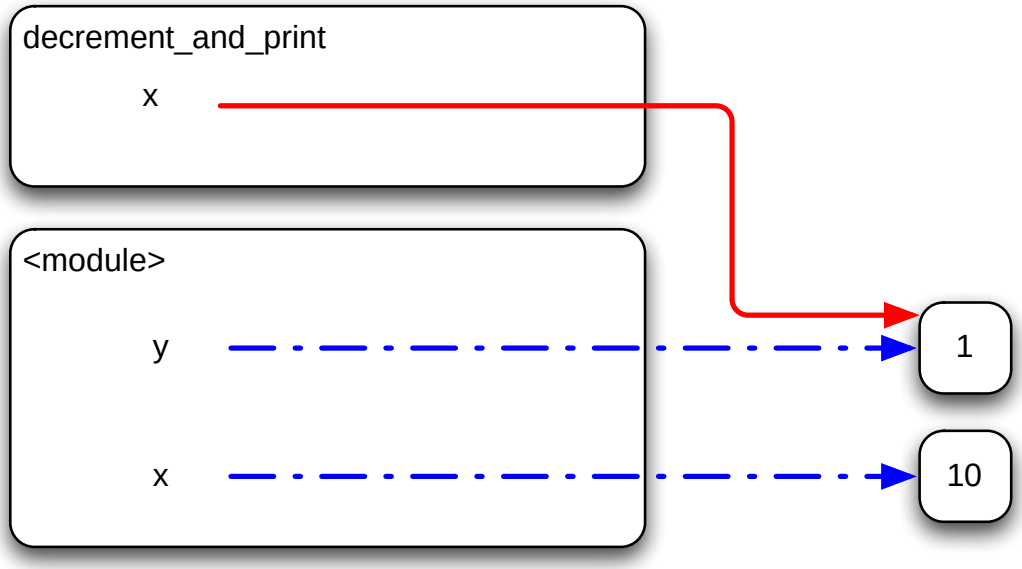
Variable and parameter scope

```
def decrement_and_print(x):
    x = x - 1
    print(x)

x = 10
y = 1
decrement_and_print(y)
```



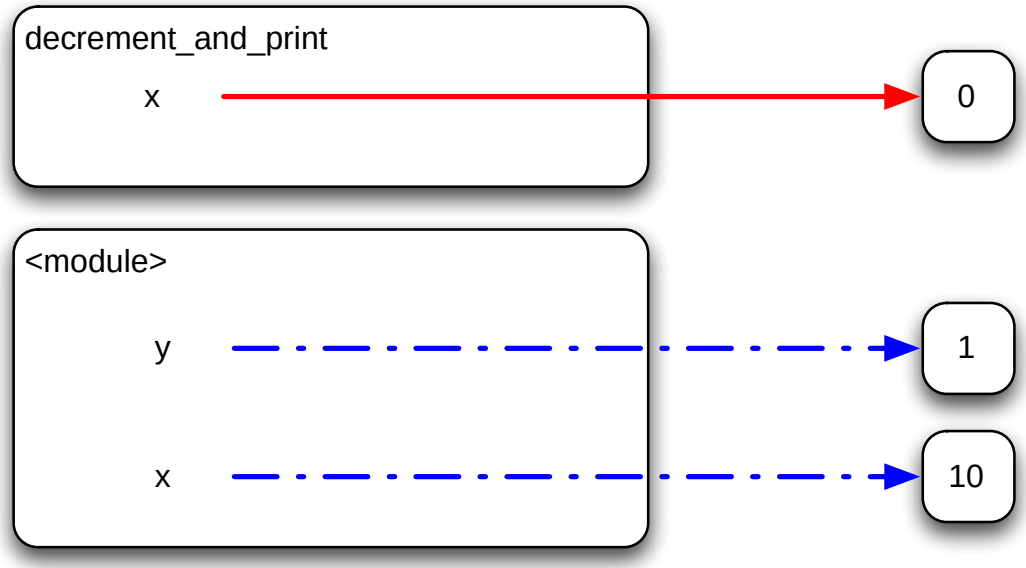
Variable and parameter scope

```
def decrement_and_print(x):
    x = x - 1
    print(x)

x = 10
y = 1
decrement_and_print(y)
```



Variable and parameter scope

```
def decrement_and_print(x):
    x = x - 1
    print(x)

x = 10
y = 1
decrement_and_print(y)
```



Functions which return values

- One of the main advantages of functions is the fact that the result of their internal processes can be returned and stored in a variable

- At a later stage, the produced value can be used to perform additional calculations

We need functions able to return values

Functions which return values

```
def power(x, y):
    return x**y

result = power(2,10)
print(result)
```

Reserved word

The result is stored in a variable

The result can be used later

In this example, another option could be:

```
print(power(2,10))
```

Functions which return values

- The function exit can be controlled

```
def divide(num, den):
    if den == 0:
        return None
    else:
        return num/den

result = divide(10, 0)
if result == None:
    print("There is a zero in the denominator")
else:
    print("The result is:", result)
```

`None` is a special value that can be used to determine if anything was wrong

Functions which return values

- More than one value can be returned

```
def max_min(x, y):
    if x > y:
        return x, y
    else:
        return y, x

mx, mn = max_min(3, 7)

print("The maximum is", mx, "and the minimum", mn)
```

The return values are separated by commas

We have to provide as many variables as return values