جامعة مولاي إسماعيل
UNIVERSITÉ MOULAY ISMAÏL

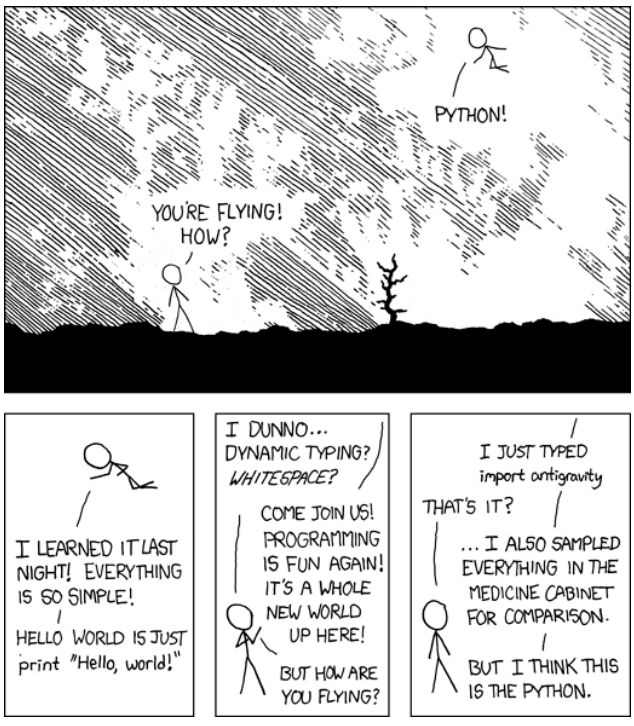
المدرسة الوطنية العليا للفنون و المهن
ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS

Python Programming
- Functions -

EL Moukhtar ZEMMOURI

ENSAM – Meknès

Version 1.0 – 2019



https://xkcd.com/353/

Function Definition

- A function definition is introduced by the keyword **def**
- Syntax and examples :

```
def function_name (formal parameters) :
     """ function's documentation string
     """
     Indented function's body
```

```
def fibo (n) :
   """ Print the first n Fibonacci numbers."""
   a, b = 0, 1
   i = 0
   while i <= n:
      print(b, end=' ')
      a, b = b, a+b
      i += 1
```

```
def is_prime (n) :
   """ Test if an positive integer is a
   prime number or not .

   Arguments :
      n (int) : a positive integer

   Returns :
      bool : True if n is prime, False otherwise
   """
   if n == 1: return False
   for i in range (2, n//2 + 1):
      if n%i == 0 : return False
   return True
```

```
>>> is_prime  #print(is_prime)
<function is_prime at 0x10c113f28>
>>> type(is_prime)
<class 'function'>
```


Function Call

- `function_name(actual parameters)`
  - Arguments are passed using call by value
  - The value is an object reference, not the value of the object
    - ➔ call by object reference
- The execution of a function allocates memory to be used for local variables
- A function without a `return` statement returns the value `None`

```
>>> fibo(10)
1 1 2 3 5 8 13 21 34 55 89
>>> is_prime(101)
True
>>> is_prime(102)
Flase
>>> primes(10)
2
3
5
…
```

```
def primes(n) :
   """ Print the first n prime numbers.

   Arguments :
      n (int) : a positive integer
   """
   p = 2; count = 0
   while count < n:
      if is_prime(p):
      count += 1
         print(p)
      p += 1
```

Default Argument Values

- Define a function that takes variable number of arguments
  - A function with arguments that have default values during definition.
  - ➔ the function can be called with fewer arguments than its definition.
- Example from built-in functions :
  - check help for `int` and `print`

Default Argument Values

- Example :

```
def triangle (c, n) :
    """ Display a triangle using char c."""
    for i in range(n):
        s = ' ' * (n-1-i) + c * (2*i + 1)
        print(s)
```

```
>>> triangle('*', 3)
    *
   ***
  *****
>>> triangle('*')
TypeError: triangle() missing 1 required
positional argument: 'n'
```

```
def triangle (c, n=5) :
    """ Display a triangle using char c."""
    for i in range(n):
        s = ' ' * (n-1-i) + c * (2*i + 1)
        print(s)
```

```
>>> triangle('*', 3)
    *
   ***
  *****
>>> triangle('*')
    *
   ***
  *****
 *******
*********
>>> triangle('+', n=3)
    +
   +++
  +++++
```

Arbitrary Argument Lists

- Define a function that can be called with an arbitrary number of arguments
  - `def function_name (*args)`
- Example :

```
def somme (*numbers) :
    """ sum up a serie of numbers."""
    s = 0
    for n in numbers:
        s += n
    return s
```

```
>>> somme(10, 20)
30
>>> somme()
0
>>> somme(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
55
```


Anonymous Functions

- Anonymous functions can be defined using `lambda` keyword
  - ➔ lambda is a keyword, never use it as an identifier ☹
- Syntax :
  - `lambda arguments : expression`
- Exemples:
  - `lambda x : x*x`
  - `lambda a, b : (a + b)/2`

```
>>> f = lambda x : x*x
>>> type(f)
<class 'function'>
>>> f(5)
25
>>> g = lambda a, b : (a+b)/2
>>> g(1, 2)
1.5
```

Recursive Functions

- Recursion is a programming paradigm
- A recursive function is a function that calls itself

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

```
def factorial (n) :
    if n==0 or n==1: return 1
    else return n* factorial(n-1)
```

```
def fibo (n) :
    a, b = 0, 1
    for i in range(1, n+1):
        a, b = b, a+b
    return b
```

```
def fibo (n) :
    if n==0 or n==1: return 1
    else return fibo(n-1) + fibo(n-2)
```

```
def fibo (n, u0, u1) :
    if n==0 : return u0
    if n==1 : return u1
    return fibo(n-1, u1, u0+u1)
```


Quick Exercise

- Implement the dichotomy (bisection) method to find zeros of a given real function
- Implement Newton's method to find zeros of a given real function