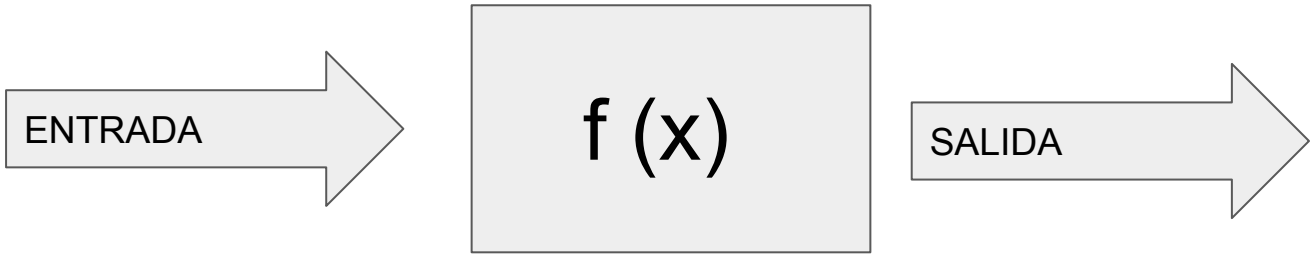
Introducción a la programación con Python
Funciones y Bucles

Alexis Rodríguez
Marcel Morán C

Esquema

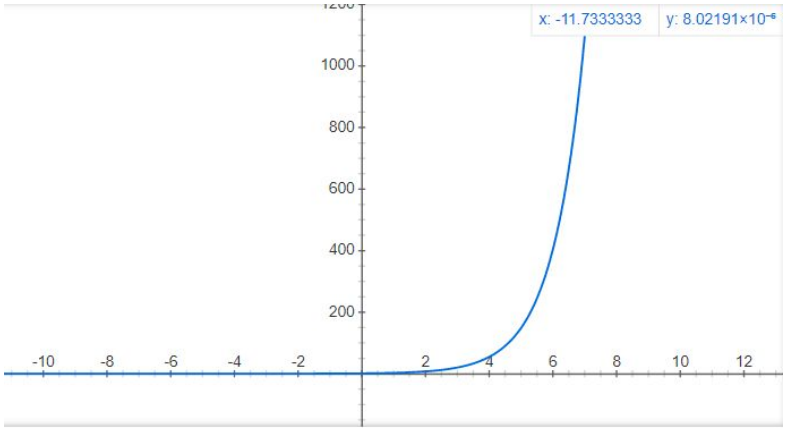
- ¿Que es una función?
- Terminología y sintaxis de una función
- Scope local vs Scope global
- El stack de llamadas
- ¿Que es un bucle?
- Terminología y sintaxis de un bucle

¿Que es una función?

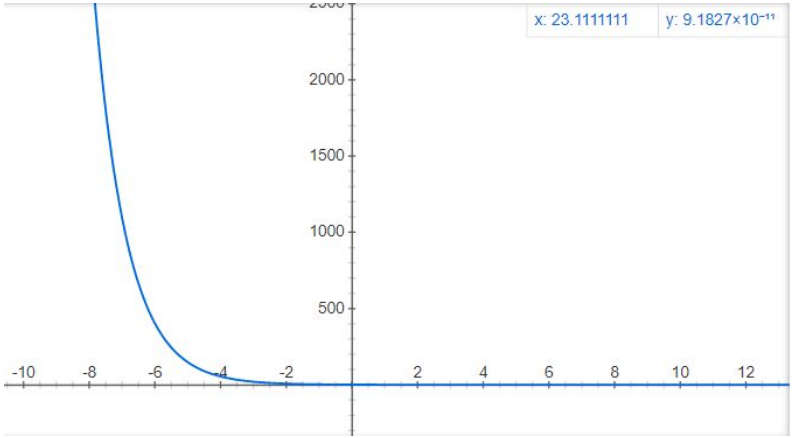


X	EXP (X)	Y
0	EXP(0)	1
1	EXP(1)	2.71
2	EXP(2)	7.4
3	EXP(3)	20.1

¿Que es una función?



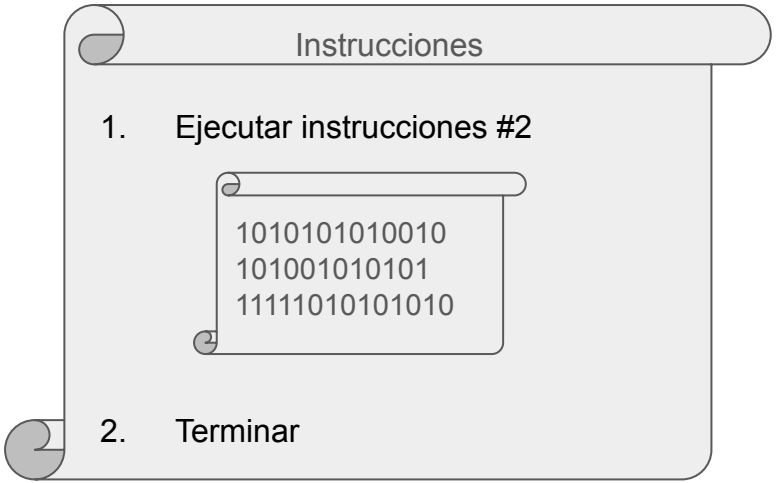
y = exp (x)



y = exp (-x)

¿Que es una función?

- Una función es como un miniprograma
- Input(), print() son ejemplos de funciones definidas en python
- Funciones pueden ser creadas
- Las funciones nos ayudan a descomponer nuestra instrucciones o código
- Facilita el reuso de código y depuración



¿Que es una función?

- Una función es como un miniprograma
- Input(), print() son ejemplos de funciones definidas en python
- Funciones pueden ser creadas
- Las funciones nos ayudan a descomponer nuestra instrucciones o código
- Facilita el reuso de código y depuración

```
>>> nombre = “Maria”
>>> print('Hola', nombre)
>>> print('Tenga un buen dia ', nombre, 'de nuevo!')
>>> print('Adios ', nombre)
```

```
Hola Maria
Tenga un buen dia Maria
Adios Maria
```

¿Que es una función?

- Una función es como un miniprograma
- Input(), print() son ejemplos de funciones definidas en python
- Funciones pueden ser creadas
- Las funciones nos ayudan a descomponer nuestra instrucciones o código
- Facilita el reuso de código y depuración

```
>>> nombre = “Maria”
>>> segundo_nombre = “Juan”
>>> print('Hola', nombre)
>>> print('Tenga un buen dia ', nombre, ' de nuevo!')
>>> print('Adios ', nombre)
>>> print('Hola', segundo_nombre )
>>> print('Tenga un buen dia ', segundo_nombre , ' de nuevo!')
>>> print('Adios ', segundo_nombre )
```

```
Hola Maria
Tenga un buen dia Maria de nuevo!
Adios Maria
Hola Juan
Tenga un buen dia Juan de nuevo!
Adios Juan
```

¿Que es una función?

- Una función es como un miniprograma
- Input(), print() son ejemplos de funciones definidas en python
- Funciones pueden ser creadas
- Las funciones nos ayudan a descomponer nuestra instrucciones o código
- Facilita el reuso de código y depuración

```
>>> primer_nombre = “Maria”
>>> segundo_nombre = “Juan”
>>> def saludar_despedirse(nombre) :
>>>    print('Hola', nombre)
>>>    print('Tenga un buen dia ', nombre, ' de nuevo!')
>>>    print('Adios ', nombre)
>>>
>>> saludar_despedirse(primer_nombre)
>>> saludar_despedirse(segundo_nombre)
```

```
Hola Maria
Tenga un buen dia Maria de nuevo!
Adios Maria
Hola Juan
Tenga un buen dia Juan de nuevo!
Adios Juan
```

Terminología y sintaxis de una función

- **Palabra def para definir una función**
- Nombre de la función
- Cuerpo
- Palabra para regresar un resultado
- El valor a regresar
- Llamar
- Argumentos
- Parametros
- Pasar

```
def es_numero_par(num):
  print(“Dentro del cuerpo”)
  return num % 2 == 0

entero = int(input(“ingrese numero”))

if(es_numero_par(entero)):
  print(“Numero es par”)
else:
  print(“Numero es par”)
```

Terminología y sintaxis de una función

- Palabra def para definir una función
- **Nombre de la función**
- Cuerpo
- Palabra para regresar un resultado
- El valor a regresar
- Llamar
- Argumentos
- Parametros
- Pasar

```
def es_numero_par(num):
  print(“Dentro del cuerpo”)
  return num % 2 == 0

entero = int(input(“ingrese numero”))

if(es_numero_par(entero)):
  print(“Numero es par”)
else:
  print(“Numero es par”)
```

Terminología y sintaxis de una función

- Palabra def para definir una función
- Nombre de la función
- **Cuerpo**
- Palabra para regresar un resultado
- El valor a regresar
- Llamar
- Argumentos
- Parametros
- Pasar

```
def es_numero_par(num):
  print(“Dentro del cuerpo”)
  return num % 2 == 0

entero = int(input(“ingrese numero”))

if(es_numero_par(entero)):
  print(“Numero es par”)
else:
  print(“Numero es par”)
```

Terminología y sintaxis de una función

- Palabra def para definir una función
- Nombre de la función
- Cuerpo
- **Palabra para regresar un resultado**
- El valor a regresar
- Llamar
- Argumentos
- Parametros
- Pasar

```
def es_numero_par(num):
  print(“Dentro del cuerpo”)
  return num % 2 == 0

entero = int(input(“ingrese numero”))

if(es_numero_par(entero)):
  print(“Numero es par”)
else:
  print(“Numero es par”)
```

Terminología y sintaxis de una función

- Palabra def para definir una función
- Nombre de la función
- Cuerpo
- Palabra para regresar un resultado
- **El valor a regresar**
- Llamar
- Argumentos
- Parametros
- Pasar

```
def es_numero_par(num):
  print(“Dentro del cuerpo”)
  return num % 2 == 0

entero = int(input(“ingrese numero”))

if(es_numero_par(entero)):
  print(“Numero es par”)
else:
  print(“Numero es par”)
```

Terminología y sintaxis de una función

- Palabra def para definir una función
- Nombre de la función
- Cuerpo
- Palabra para regresar un resultado
- El valor a regresar
- **Llamar**
- Argumentos
- Parametros
- Pasar

```
def es_numero_par(num):
  print(“Dentro del cuerpo”)
  return num % 2 == 0

entero = int(input(“ingrese numero”))

if(es_numero_par(entero)):
  print(“Numero es par”)
else:
  print(“Numero es par”)
```

Terminología y sintaxis de una función

- Palabra def para definir una función
- Nombre de la función
- Cuerpo
- Palabra para regresar un resultado
- El valor a regresar
- Llamar
- **Argumentos**
- **Parametros**
- **Pasar**

```
def es_numero_par(num):
  print("Dentro del cuerpo")
  return num % 2 == 0

entero = int(input("ingrese numero"))

if(es_numero_par(entero)):
  print("Numero es par")
else:
  print("Numero es par")
```

None value

- Tipo de dato None
- Especifica la ausencia de un valor
- Se usa para evitar que algo se confunda con una variable real
- Cualquier función que no especifique la instrucción return regresa un None

```
contenedor = print('Hola')
>>> 'Hola'
contenedor == None
>>> True
```

0 | NONE

Scope local vs Scope global

- Scope local : Parámetros y variables en una función llamada viven dentro de la función
- Scope global: Variables asignadas fuera la función
- Variables tienen que globales o locales
- Código en el scope global no puede usar variables locales
- Código en scope local puede usar variables globales
- Código de scope local de una función no puede ser variables de otros scope locales
- Variables pueden usar el mismo símbolo si viven en diferentes scopes

Scope local vs Scope global

- Scope local : Parámetros y variables en una función llamada viven dentro de la función
- Scope global: Variables asignadas fuera la función
- **Variables tienen que globales o locales**
- **Código en el scope global no puede usar variables locales**
- **Código en scope local puede usar variables globales**
- **Código de scope local de una función no puede ser variables de otros scope locales**
- **Variables pueden usar el mismo símbolo si viven en diferentes scopes**

```
def funct_cambiar():
  print('funcion llamada')
  variable_local = 0

variable_local = 10
print(variable_local)
funct_cambiar()
print(variable_local)
10
funcion llamada
10
```

Scope local vs Scope global

- Scope local : Parámetros y variables en una función llamada viven dentro de la función
- Scope global: Variables asignadas fuera la función
- **Variables tienen que globales o locales**
- **Código en el scope global no puede usar variables locales**
- **Código en scope local puede usar variables globales**
- **Código de scope local de una función no puede ser variables de otros scope locales**
- **Variables pueden usar el mismo símbolo si viven en diferentes scopes**

```
def funct_cambiar():
  global variable_local
  print('funcion llamada')
  variable_local = 0

variable_local = 10
print(variable_local)
funct_cambiar()
print(variable_local)
10
funcion llamada
0
```

Scope local vs Scope global

- Scope local : Parámetros y variables en una función llamada viven dentro de la función
- Scope global: Variables asignadas fuera la función
- **Variables tienen que globales o locales**
- **Código en el scope global no puede usar variables locales**
- **Código en scope local puede usar variables globales**
- **Código de scope local de una función no puede ser variables de otros scope locales**
- **Variables pueden usar el mismo símbolo si viven en diferentes scopes**

```
def funct_cambiar():
  variable_local = 0

def funct_referencia():
  ref_var_local = variable_local

 funct_cambiar()
 funct_referencia()
 val = variable_local
```

Scope local vs Scope global

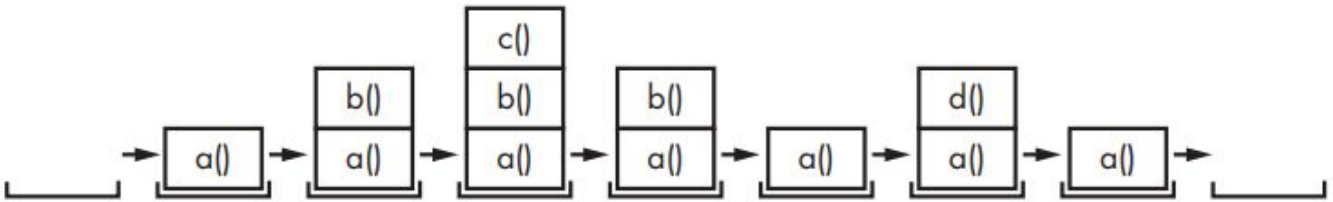
- Scope local : Parámetros y variables en una función llamada viven dentro de la función
- Scope global: Variables asignadas fuera la función
- **Variables tienen que globales o locales**
- **Código en el scope global no puede usar variables locales**
- **Código en scope local puede usar variables globales**
- **Código de scope local de una función no puede ser variables de otros scope locales**
- **Variables pueden usar el mismo símbolo si viven en diferentes scopes**

```
def funct_cambiar():
  variable_local = 0

def funct_referencia():
  variable_local = 5


 variable_local = 10
```

La pila(Stack) de llamadas

- Stack es un remesas de hojas
- Llamadas crean un frame que se añade a la pila de llamadas
- Funciones pueden llamar otras funciones

```
def a():
  b()
  d()
def b():
  c()
def c():
  pass
def d():
  pass

a()
```



¿Que es un bucle?

Buenos dias

```
>>> nombre = input('Cual es tu nombre?: ')
>>> print('Hola', nombre)
>>> print('Hola', nombre, 'de nuevo!')
>>> print('Hola', nombre, 'de nuevo!')
```

Buenos dias

```
Hola Marcel
Hola Marcel de nuevo!
Hola Marcel de nuevo!
```

Buenos dias

```
>>> nombre = input('Cual es tu nombre?: ')
>>> print('Hola', nombre)
>>> print('Hola', nombre, 'de nuevo!')
>>> print('Hola', nombre, 'de nuevo!')
>>> print('Hola', nombre, 'de nuevo!')
>>> print('Hola', nombre, 'de nuevo!')
```

Buenos dias

```
Hola Marcel
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
```

Buenos dias

```
Hola Marcel
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
```

Terminología y sintaxis de un bucle

- **`for loop`**

  Nos permite ejecutar un bloque de código tantas veces como sean especificadas con anticipacion.

```
for <variable> in range(<algun_numero>):
      <expresion>
      <expresion>
      . . .
```

Opcionales

```
range(inicio,fin,aumento)
```

Nota: fin no esta incluido

Ejemplos:

```
valores = ''
for indice in range(10):
   valores += str(indice) + ' '
print('Valores:', valores)
```

```
Valores: 0 1 2 3 4 5 6 7 8 9
```

```
valores = ''
for indice in range(5, 10):
   valores += str(indice) + ' '
print('Valores:', valores)
```

```
Valores: 5 6 7 8 9
```

```
valores = ''
for indice in range(5, 10, 2):
   valores += str(indice) + ' '
print('Valores:', valores)
```

```
Valores: 5 7 9
```

Terminología y sintaxis de un bucle

```
nombre = input('Cual es tu nombre?')
repeticiones = 10
for index in range(repeticiones):
    if index == 0:
        print('Hola', nombre)
    else:
        print('Hola', nombre, 'de nuevo!')
```

Buenos dias

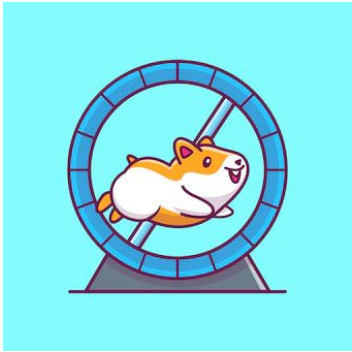
Terminología y sintaxis de un bucle

- **`while` loop**

  Nos permite ejecutar un bloque de código mientras una condición sea verdadera.

```
while <condicion>:
      <expresion>
      <expresion>
      . . .
```

Boolean



Terminología y sintaxis de un bucle

- **while loop**

```
nombre = input('Cual es tu nombre?')
repeticion = 0
while repeticion < 10:
   if repeticion == 0:
       print('Hola', nombre)
   else:
       print('Hola', nombre, 'de nuevo!')
   repeticion += 1
```

```
Hola Marcel
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
Hola Marcel de nuevo!
```

Terminología y sintaxis de un bucle

- **while loop**

```
acumulador = 0
while True:
    operador = input('Operador: ')
    valor = float(input('Valor: '))
    if operador == '+':
        acumulador += valor
    elif operador == '-':
        acumulador -= valor
    elif operador == '*':
        acumulador *= valor
    elif operador == '/':
        acumulador /= valor
    else:
        print('Operador invalido! Terminando el programa')
        print('Resultado: ', acumulador)
        break
```

```
Operador invalido!
Terminando el programa
Resultado:  -4.0
```

Conclusión

- Una función es como un mini programa, facilita el reuso de código y la depuración
- Una función se crea con la instrucción def nombre_fun() :
- Una función puede regresar un valor con la instrucción return
- Parámetros no son lo mismo que argumentos
- None es un tipo de dato que especifica la ausencia de un valor
- Toda función que no especifique return, regresara un tipo None
- Las reglas de local scope y global scope deben ser respetadas para evitar problemas
- La pila de llamadas guarda un cuadro(frame) cuando una función es llamada
- Un bloque facilita el reuso de código y la depuración