Programación de Computadores: Introducción a Python

Juan F. Pérez

Departamento MACC
Matemáticas Aplicadas y Ciencias de la Computación
Universidad del Rosario

juanferna.perez@urosario.edu.co

Segundo Semestre de 2017

Contenidos

Introducción

¿Qué es Python?

Respuestas...

¿Qué es Python?

Respuestas...

- Un lenguaje de programación...

¿Qué es Python?

Respuestas...

- Un lenguaje de programación...
- ... de alto nivel.

¿Qué es Python?

Respuestas...

- Un lenguaje de programación...
- ... de alto nivel.
- ... imperativo.

Algunas características de Python

Algunas características de Python

- Requiere pocas líneas de código comparado con otros lenguajes (Java o C++)

Algunas características de Python

- Requiere pocas líneas de código comparado con otros lenguajes (Java o C++)
- Usa indentación (sangrado) para separar bloques de código

Algunas características de Python

- Requiere pocas líneas de código comparado con otros lenguajes (Java o C++)
- Usa indentación (sangrado) para separar bloques de código
- Simple pero poderoso, usado en proyectos de gran escala

Algunas características de Python

- Requiere pocas líneas de código comparado con otros lenguajes (Java o C++)
- Usa indentación (sangrado) para separar bloques de código
- Simple pero poderoso, usado en proyectos de gran escala
- Usado en muchas áreas científicas (e.g., aprendizaje de máquina)

Algo de historia

- Desarrollado inicialmente por Guido Van Rossum en 1989, inspirado en ABC, un lenguaje usado en CWI (Centrum Wiskunde en Informatica, Amsterdam)

Algo de historia

- Desarrollado inicialmente por Guido Van Rossum en 1989, inspirado en ABC, un lenguaje usado en CWI (Centrum Wiskunde en Informatica, Amsterdam)
- Guido sobre Python: "in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus)."

Algo de historia

- Desarrollado inicialmente por Guido Van Rossum en 1989, inspirado en ABC, un lenguaje usado en CWI (Centrum Wiskunde en Informatica, Amsterdam)
- Guido sobre Python: "in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus)."
- `https://en.wikipedia.org/wiki/Python_(programming_language)`

Algo más de historia

- Primer desarrollo en 1989

Algo más de historia

- Primer desarrollo en 1989
- Python 2.0: lanzado en 2000

Algo más de historia

- Primer desarrollo en 1989
- Python 2.0: lanzado en 2000
- Python 3.0: lanzado en 2008

Algo más de historia

- Primer desarrollo en 1989
- Python 2.0: lanzado en 2000
- Python 3.0: lanzado en 2008
  - No es compatible con la versión 2.X
  - Muchos paquetes aún funcionan con la versión 2.X

Algo más de historia

- Primer desarrollo en 1989
- Python 2.0: lanzado en 2000
- Python 3.0: lanzado en 2008
  - No es compatible con la versión 2.X
  - Muchos paquetes aún funcionan con la versión 2.X
- Nosotros trabajaremos con la versión 2.X (2.7) pero las diferencias con las 3.X son mínimas en nuestro código.

Un primer código

El lenguaje Python

- Algunas instrucciones primitivas
- Algo de gramática
- Algunos símbolos/palabras reservados

Imprimiendo en Python

```
print "Hola Mundo!"
```

¿Cómo ejecutamos este código?

Windows (línea de comandos):

- Creemos una carpeta cuya ubicación llamaremos *path*
- Escribimos el programa en un archivo "miEjemplo1.py"
- Inicio + `cmd`
- Ejecutamos "python –version" para conocer la versión de python instalada
- Vamos a la carpeta donde guardamos el archivo (`cd` path)
- Ejecutamos "python miEjemplo1.py" (escribir esta línea + Enter)

¿Cómo ejecutamos este código?

Windows (IDLE):

- Inicio+IDLE
- File -¿ New File
- Escribimos el programa y lo guardamos como un archivo "miEjemplo1.py"
- F5
- Resultado en IDLE Shell

¿Cómo ejecutamos este código?

Linux (línea de comandos):

- Creemos una carpeta cuya ubicación llamaremos *path*
- Escribimos el programa en un archivo "miEjemplo1.py"
- `ctrl+t` para abrir una terminal
- Ejecutamos "python –version" para conocer la versión de python instalada
- Vamos a la carpeta donde guardamos el archivo (cd *path*)
- Ejecutamos "python miEjemplo1.py"

Imprimiendo texto

Imprimiendo en Python

```
print "Hola Mundo!"
print "Este es mi primer programa en Python"
```

Imprimiendo en Python

```
print "Hola Mundo!"
print "Este es mi primer programa en Python"
print 'Este es mi primer programa en Python'
```

Imprimiendo en Python

```
print "Hola Mundo!"
print "Este es mi primer programa en Python"
print 'Este es mi primer programa en Python'
print 'Aunque hace "un poco" de calor'
```

Imprimiendo en Python

```
print "Hola Mundo!"
print "Este es mi primer programa en Python"
print 'Este es mi primer programa en Python'
print 'Aunque hace "un poco" de calor'
```

- `print`: instrucción
- Imprime lo que esté entre comillas (dobles o simples)
- Usa el mismo tipo de comillas al inicio y al final

Agregar comentarios

Símbolo: #

```
print "Hola Mundo!"
# print "Esto no lo quiero imprimir"
print "Esto si lo quiero imprimir"
```

Agregar comentarios

Símbolo: #

```
print "Hola Mundo!"
# print "Esto no lo quiero imprimir"
print "Esto si lo quiero imprimir"
```

¿Que tal si intentamos imprimir?

```
print "Esto sí lo quiero imprimir"
```

Agregar comentarios

Símbolo: #

```
print "Hola Mundo!"
# print "Esto no lo quiero imprimir"
print "Esto sí lo quiero imprimir"
```

Agregar comentarios

Símbolo: #

```
print "Hola Mundo!"
# print "Esto no lo quiero imprimir"
print "Esto sí lo quiero imprimir"
```

¿Cómo incluir tildes y otros símbolos?

La primera línea de código debe ser

```
# -*- coding: utf-8 -*-
```

Codificación de texto UTF-8

Agregar comentarios

Símbolo: #

```
# -*- coding: utf-8 -*-
print "Hola Mundo!"
# print "Esto no lo quiero imprimir"
print u"Esto sí lo quiero imprimir"
```

Operaciones aritméticas

Operaciones aritméticas

Símbolos:

- $+, -, *, /, \%$
- $<, >, =$

```
print "En esta sala hay:"
print "Computadores: ", 13
print "Sillas: ", 12*2 + 1
print "Estudiantes:", 3 + 2 + 5 - 2
print "Ventanas:", 3 + 2*10 + 25/5
```

Operaciones aritméticas

Símbolos:

- $+, -, *, /, \%$
- $<, >, =$

```
print "En esta sala hay:"
print "Computadores: ", 13
print "Sillas: ", 12*2 + 1
print "Estudiantes:", 3 + 2 + 5 - 2
print "Ventanas:", 3 + 2*10 + 25/5

print "Sillas por computador:", 25/13
print "Sillas que sobran: ", 25%13
```

Operaciones aritméticas

Símbolos:

- $+, -, *, /, \%$
- $<, >, =$

```
print "En esta sala hay:"
print "Computadores: ", 13
print "Sillas: ", 12*2 + 1
print "Estudiantes:", 3 + 2 + 5 - 2
print "Ventanas:", 3 + 2*10 + 25/5

print "Sillas por computador:", 25/13
print "Sillas que sobran: ", 25%13
```

Intentar...

```
print "Sillas por computador:", 26/13
print "Sillas que sobran: ", 26%13
```

Operaciones aritméticas - precedencia

Precedencia:

1. Unaria: $-$ en -1

Operaciones aritméticas - precedencia

Precedencia:

1. Unaria: $-$ en -1
2. Exponentes y raíces: a^b, $\sqrt{a}$

Operaciones aritméticas - precedencia

Precedencia:

1. Unaria: $-$ en -1
2. Exponentes y raíces: a^b, $\sqrt{a}$
3. Productos y divisiones: $a * b$, a/b, $a\,\%\,b$

Operaciones aritméticas - precedencia

Precedencia:

1. Unaria: $-$ en -1
2. Exponentes y raíces: a^b, $\sqrt{a}$
3. Productos y divisiones: $a * b$, a/b, $a\,\%\,b$
4. Sumas y restas: $a + b$, $a - b$

Operaciones aritméticas - precedencia

Precedencia:

1. Unaria: $-$ en -1
2. Exponentes y raíces: a^b, $\sqrt{a}$
3. Productos y divisiones: $a * b$, a/b, $a\%b$
4. Sumas y restas: $a + b$, $a - b$
5. PEMDAS: Parenthesis, Exponents, Multiplications, Divisions, Additions, Substractions

Evaluar una desigualdad

```
print "Es 3*7 > 2*10 ? "
print 3*7 > 2*10
```

Evaluar una desigualdad

```
print "Es 3*7 > 2*10 ? "
print 3*7 > 2*10

print "Es 3*7 < 2*10+1 ? "
print 3*7 < 2*10
```

Evaluar una desigualdad

```
print "Es 3*7 > 2*10 ? "
print 3*7 > 2*10

print "Es 3*7 < 2*10+1 ? "
print 3*7 < 2*10

print "Es 3*7 <= 2*10+1 ? "
print 3*7 <= 2*10 +1
```

Enteros vs. punto flotante

```
print "33/7 = ", 33/7
print "33.0/7.0 = ", 33.0/7.0
```

Variables

Variables

```
gatos = 5.0
personas = 3
print "en mi casa hay ", gatos, "gatos"
print "en mi casa vivimos ", personas, "personas"
print "hay ", gatos/personas, "gatos por persona"
```

Variables

```
gatos = 5.0
personas = 3
print "en mi casa hay ", gatos, "gatos"
print "en mi casa vivimos ", personas, "personas"
print "hay ", gatos/personas, "gatos por persona"
```

Intentar...

```
print "en mi casa hay ", perros, "perros"
```

Variables

- Variables: ubicación/dirección en memoria + identificador

Variables

- Variables: ubicación/dirección en memoria + identificador
- Identificador: nombre, permite manipular la variable, usar y alterar su valor

Variables

- Variables: ubicación/dirección en memoria + identificador
- Identificador: nombre, permite manipular la variable, usar y alterar su valor
- Ubicación en memoria: guarda/contiene el valor asignado a la variable

Variables

- Variables: ubicación/dirección en memoria + identificador
- Identificador: nombre, permite manipular la variable, usar y alterar su valor
- Ubicación en memoria: guarda/contiene el valor asignado a la variable

Ejemplo

```
personas = 3
```

- Identificador: `personas`
- Asignamos el valor 3 la variable personas

Variables

- Identificador permanece *fijo* durante la ejecución del programa
- Valor asignado *puede cambiar* durante la ejecución del programa

Variables

- Identificador permanece *fijo* durante la ejecución del programa
- Valor asignado *puede cambiar* durante la ejecución del programa

Ejemplo

```
residentes = 3
personas = 3
print "En casa viven", personas, "personas"
invitados = 4
personas = residentes + invitados
print "Pero hoy hay", personas, "personas"
```

Tipos de Variables

- Variables pueden ser diferentes tipos

Tipos de Variables

- Variables pueden ser diferentes tipos
- Algunos tipos *en python* incluyen:
  - `str` (string): cadenas de caracteres
  - `int` (integer): enteros
  - `float` (floating point): números de punto flotante (representan números reales)

Tipos de Variables

- Variables pueden ser diferentes tipos
- Algunos tipos *en python* incluyen:
  - `str` (string): cadenas de caracteres
  - `int` (integer): enteros
  - `float` (floating point): números de punto flotante (representan números reales)

Ejemplo

```
gatos = 5.0
personas = 3
mensaje = "gastos y personas:"
print gatos
print personas
print mensaje
```

Tipos de Variables

- Python no requiere que especifiquemos el tipo de variable (otros lenguajes sí)

Tipos de Variables

- Python no requiere que especifiquemos el tipo de variable (otros lenguajes sí)
  - Al definir `gatos = 5.0`
  - Python define `gatos` como una variable de tipo `float`

Tipos de Variables

- Python no requiere que especifiquemos el tipo de variable (otros lenguajes sí)
  - Al definir `gatos = 5.0`
  - Python define `gatos` como una variable de tipo `float`
  - Al definir `personas = 3`
  - Python define `personas` como una variable de tipo `int`

Tipos de Variables

- Python no requiere que especifiquemos el tipo de variable (otros lenguajes sí)
  - Al definir `gatos = 5.0`
  - Python define `gatos` como una variable de tipo `float`
  - Al definir `personas = 3`
  - Python define `personas` como una variable de tipo `int`
  - Al definir `mensaje = "gastos y personas:"`
  - Python define `mensaje` como una variable de tipo `str`

Variables de tipo `float`

- Representan números reales

Variables de tipo `float`

- Representan números reales
  - $$r = c \times b^e$$

Variables de tipo `float`

- Representan números reales
  - $$r = c \times b^e$$
  - r: número real

Variables de tipo `float`

- Representan números reales
  - $$r = c \times b^e$$
  - r: número real
  - c: coeficiente

Variables de tipo `float`

- Representan números reales
  - $$r = c \times b^e$$
  - r: número real
  - c: coeficiente
  - b: base

Variables de tipo `float`

- Representan números reales
  - $$r = c \times b^e$$
  - r: número real
  - c: coeficiente
  - b: base
  - e: exponente

Variables de tipo `float`

- Representan números reales
  - $$r = c \times b^e$$
  - r: número real
  - c: coeficiente
  - b: base
  - e: exponente
  - Ejemplos:

$$0,23 = 2,3 \times 10^{-1}$$

$$-243,2 = -2,432 \times 10^{2}$$

Variables de tipo `float`

- Representan números reales
  - $$r = c \times b^e$$
  - r: número real
  - c: coeficiente
  - b: base
  - e: exponente
  - Ejemplos:
    $$0,23 = 2,3 \times 10^{-1}$$
    $$-243,2 = -2,432 \times 10^2$$
  - Número de dígitos en la representación del coeficiente: número de cifras significativas

Más Tipos de Variables: `boolean`

Booleanas: toman valor falso o verdadero

```
cond1 = 5 < 10
print "Es 5 < 10?", cond1
cond2 = 20 < 10
print "Es 20 < 10?", cond2
```

Más Tipos de Variables: `boolean`

Booleanas: toman valor falso o verdadero

```
cond1 = 5 < 10
print "Es 5 < 10?", cond1
cond2 = 20 < 10
print "Es 20 < 10?", cond2
```

Operaciones Booleanas: `and`, `or`, `not`

```
print "Es 5 < 10 y 20 < 10?", cond1 and cond2
print "Es 5 < 10 o 20 < 10?", cond1 or cond2
print "Es 5 no menor que 10?", not cond1
```

Otras comparaciones numéricas

Igual

```
print "Es 5 = 10?", cond1
cond2 = 2*5 == 10
print "Es 2*5 = 10?", cond2
cond3 = 10.0 == 10
print "Es 10.0 = 10?", cond3
```

Otras comparaciones numéricas

Igual

```
print "Es 5 = 10?", cond1
cond2 = 2*5 == 10
print "Es 2*5 = 10?", cond2
cond3 = 10.0 == 10
print "Es 10.0 = 10?", cond3
```

No igual

```
cond1 = 5 != 10
print "Es 5 != 10?", cond1
cond2 = 2*5 != 10
print "Es 2*5 != 10?", cond2
cond3 = 10.0 != 10
print "Es 10.0 != 10?", cond3
```

Otros tipos de variables

- `long`: enteros grandes no representables con `int`
- `complex`: números complejos

Texto con formato

Texto con formato

Podemos usar variables al imprimir un texto

```
resid = 3
invit = 4
personas = resid + invit
print "En casa hay %d personas en total" %personas
print "%d residentes y %d invitados" %(resid, invit)
```

Texto con formato: string, enteros y flotantes

```
resid = 3
invit = 4.0
invit_por_resid = invit/resid
mensaje = "Bienvenido"
print "Cada uno de los %d residentes tiene \
a cargo %f invitados" %(resid, invit_por_resid)
print "Imprimimos tarjetas con el \
mensaje %s para cada invitado " %mensaje
```

El operador \

Nos permite dividir una línea de comandos en varias líneas.

Texto con formato: string, enteros y flotantes

Los símbolos determinan el tipo de dato a imprimir:

- **% d**: entero
- **% f**: flotante
- **% s**: string

Texto con formato: string, enteros y flotantes

Los símbolos determinan el tipo de dato a imprimir:

- **% d**: entero
- **% f**: flotante
- **% s**: string

Intentar...

```
mensaje = "Bienvenido"
print "Mi mensaje: %d" %mensaje
```

Texto con formato: string, enteros y flotantes

Texto con formato sin variables

```
print "Cada uno de los %d residentes \
tiene a cargo %f invitados" %(3, 4.0/3)
print "Imprimimos tarjetas con el mensaje %s \
para cada invitado " %"Bienvenido"
```

Operaciones con cadenas de caracteres

```
mensaje = "Bienvenido"
nombre = "Camilo"
print mensaje * 3
print mensaje + nombre
```

Operaciones con cadenas de caracteres

```
mensaje = "Bienvenido"
nombre = "Camilo"
print mensaje * 3
print mensaje + nombre
```

```
print mensaje + " " + nombre + "!"
```

Operaciones con cadenas de caracteres

```
mensaje = "Bienvenido"
nombre = "Camilo"
print mensaje * 3
print mensaje + nombre
```

```
print mensaje + " " + nombre + "!"
```

```
print (mensaje + " " + nombre + "! ") * 3
```

Representación %r

```
mensaje = "%r-%r-%r-%r" %(1, 3.14, "hola", True)
print mensaje
mensaje = "%d-%f-%s-%s" %(1, 3.14, "hola", True)
print mensaje
```

Representación % r

```
mensaje = "%r-%r-%r-%r" %(1, 3.14, "hola", True)
print mensaje
mensaje = "%d-%f-%s-%s" %(1, 3.14, "hola", True)
print mensaje
```

- **% r** devuelve la representación de la variable

Representación % r

```
mensaje = "%r - %r - %r - %r" %(1, 3.14, "hola", True)
print mensaje
mensaje = "%d - %f - %s - %s" %(1, 3.14, "hola", True)
print mensaje
```

- **% r** devuelve la representación de la variable
- Útil para debugging, no para imprimir resultados

Representación % r

```
mensaje = "%r - %r - %r - %r" % (1, 3.14, "hola", True)
print mensaje
mensaje = "%d - %f - %s - %s" % (1, 3.14, "hola", True)
print mensaje
```

- **% r** devuelve la representación de la variable
- Útil para debugging, no para imprimir resultados
- Para booleanos, usar %s

Saltos de línea y tabuladores

```
lista1 = u"Cálculo, Lógica, Programación"
lista2 = u"Cálculo\nLógica\nProgramación"
lista3 = u"Cálculo\tLógica\tProgramación"
print "Estoy viendo %s" % lista1
print "Estoy viendo %s" % lista2
print "Estoy viendo %s" % lista3
```

Comandos y secuencias de escape

Comando de escape: permite incluir información diferente al contexto actual

Comandos y secuencias de escape

Comando de escape: permite incluir información diferente al contexto actual

- \ : comando de escape en Python
- \n : nueva línea
- \t : tabulador

Comandos y secuencias de escape

Comando de escape: permite incluir información diferente al contexto actual

- \ : comando de escape en Python
- \n : nueva línea
- \t : tabulador
- \" : comillas

Comandos y secuencias de escape

Comando de escape: permite incluir información diferente al contexto actual

- \ : comando de escape en Python
- \n : nueva línea
- \t : tabulador
- \" : comillas
- \\ : backslash

Comandos y secuencias de escape

Comando de escape: permite incluir información diferente al contexto actual

- \ : comando de escape en Python
- \n : nueva línea
- \t : tabulador
- \" : comillas
- \\ : backslash

```
print "\\_/"
print "\\_//"
print "\\_//\\_//"
print u"Le dije \"hola\" pero no me respondió"
```

Imprimir múltiples líneas

```
mensaje = """
Hola,
Hoy tengo ganas de salir a caminar.
Que tal si nos vemos a las 9?
Saludos,
Yo
"""
print mensaje
```

Funciones: Definiendo nuevas instrucciones

Funciones

- No queremos tener todo el código en un solo bloque
- Queremos definir en un solo sitio acciones repetitivas: *funciones* (nuevas instrucciones para Karel)
- Y llamamos estas funciones/instrucciones cada vez que las necesitemos

Funciones en Python

Para Karel

- `DEFINE-NEW-INSTRUCTION` miNuevaInstruc `AS`
- `BEGIN`
- `...`
- `END`

Funciones en Python

Para Karel

- `DEFINE-NEW-INSTRUCTION` miNuevaInstruc `AS`
- `BEGIN`
- ...
- `END`

En Python

```
def miNuevaInstruc():
    ...
```

Funciones en Python - Ejemplo

```
def imprimirSeparador():
    print "- - - - - - - - - - - - - - -"
    print ""
    print "- - - - - - - - - - - - - - -"

color1 = "Amarillo"
color2 = "Verde"
color3 = "Rojo"
print color1
imprimirSeparador()
print color2
imprimirSeparador()
print color3
```

Funciones en Python

- Todas las funciones se definen al principio (como para Karel)

Funciones en Python

- Todas las funciones se definen al principio (como para Karel)
- La definición de una función sigue el formato

```
def nombre():
    instruccion 1
    instruccion 2
    instruccion 3
    ...
```

Funciones en Python

- Todas las funciones se definen al principio (como para Karel)
- La definición de una función sigue el formato

```
def nombre():
    instruccion 1
    instruccion 2
    instruccion 3
    ...
```

- Note la indentación (4 espacios estándar - *evite tabs*)

Funciones en Python

- Todas las funciones se definen al principio (como para Karel)
- La definición de una función sigue el formato

```
def nombre():
    instruccion 1
    instruccion 2
    instruccion 3
    ...
```

- Note la indentación (4 espacios estándar - *evite tabs*)
- Después de definidas, llamamos las funciones como

```
nombre()
```

Funciones en Python - Otro Ejemplo

```
# -*- coding: utf-8 -*-
def imprimeDia():
    print "15"
def imprimeMes():
    print "Enero"

print u"¿Qué día cumples años?"
imprimeDia()
print u"¿En qué mes?"
imprimeMes()
```

Funciones: argumentos y resultados

Funciones: pasando argumentos

- Las funciones no pueden cambiar su definición
- Pero pueden recibir argumentos que cambian su resultado (o incluso las instrucciones que ejecuta)

```
# -*- coding: utf-8 -*-
def imprimaNumero(arg1):
    print u"El número es %d" % arg1
    print "_ _ _ _ _ _"

imprimaNumero(1)
imprimaNumero(10)
imprimaNumero(100)
imprimaNumero(1000)
```

Funciones con argumentos: otro ejemplo

```
# -*- coding: utf-8 -*-
def imprimaSuma(arg1,arg2):
    print u"La suma %f + %f es igual a %f" \
                %(arg1,arg2,arg1+arg2)
    print "------"
def imprimaProducto(arg1,arg2):
    print u"El producto %f * %f es %f" \
                %(arg1,arg2,arg1*arg2)
    print "------"

imprimaSuma(2, 5)
imprimaProducto(2, 5)
imprimaSuma(10.5, 5.2)
imprimaProducto(10.5, 5.2)
```

Funciones con argumentos: y otro ejemplo

```
# -*- coding: utf-8 -*-
def imprimaDiferencia(nota1, nota2):
    print nota1
    print nota2
    print nota1-nota2

miNota = 95;
tuNota = 98;
imprimaDiferencia(miNota, tuNota)
imprimaDiferencia(miNota, 90)
imprimaDiferencia(tuNota+1, miNota-10)
print "mi nota: %d" %miNota
print "tu nota: %d" %tuNota
```

Funciones: retornando resultados

- Las funciones pueden retornar un resultado
- Valor del resultados se puede usar en el programa principal (o la función que llama a la otra función)

Funciones: retornando resultados

- Las funciones pueden retornar un resultado
- Valor del resultados se puede usar en el programa principal (o la función que llama a la otra función)

```
def suma(a,b):
        c = a + b
        return c

num1 = 5
num2 = 63
num3 = 18
res1 = suma(num1,num2)
print "%f + %f = %f" %(num1, num2, res1)
res2 = suma(res1,num3)
print "%f + %f = %f" %(res1, num3, res2)
```

Funciones: retornando varios resultados

```
# -*- coding: utf-8 -*-
def sumaDif(a,b):
        suma = a + b
        dif = a - b
        return suma, dif

num1 = 5
num2 = 63
suma12, dif12 = sumaDif(num1,num2)
print u"Números: %f, %f" %(num1, num2)
print u"Suma: %f" %suma12
print u"Diferencia: %f" %dif12
```

Más funciones (incluidas)

Funciones incluidas (built-in) en Python

- Python incluye muchas funciones por defecto: `abs()`

Funciones incluidas (built-in) en Python

- Python incluye muchas funciones por defecto: `abs()`

```
# -*- coding: utf-8 -*-
def sumaDif(a,b):
        suma = a + b
        dif = abs(a - b)
        return suma, dif

num1 = 5
num2 = 63
suma12, dif12 = sumaDif(num1,num2)
print u"Números: %f, %f" %(num1, num2)
print u"Suma: %f" %suma12
print u"Diferencia absoluta: %f" %dif12
```

Funciones para conversión de tipos (casting)

- Cambiamos el tipo de un dato a través de una función

Funciones para conversión de tipos (casting)

- Cambiamos el tipo de un dato a través de una función

```
# -*- coding: utf-8 -*-
num1 = 5.2
print u"Número original: %f" %num1
print u"Número convertido a entero: %d" %int(num1)
print u"Número convertido a string: %s" %str(num1)
str1 = "63"
print u"String original: %s" %str1
print u"String convertido a entero: %d" %int(str1)
print u"String convertido a flotante: %f" %float(str1)
str2 = "63.5"
print u"String original: %s" %str2
print u"String convertido a flotante: %f" %float(str2)
```

Funciones para leer input del usuario

- Solicitamos información del usuario
- `raw_input()` lee la respuesta y la retorna en forma de string

Funciones para leer input del usuario

- Solicitamos información del usuario
- `raw_input()` lee la respuesta y la retorna en forma de string

```
# -*- coding: utf-8 -*-
print u"¿Edad?"
edad = raw_input()
print u"¿Ocupación?"
ocupacion = raw_input()
print u"- - - - - - - - - -"
print u"Información:"
print u"Edad: %s" %edad
print u"Ocupación: %s" %ocupacion
```

Operando con input del usuario

- Usamos `raw_input()` para obtener información del usuario
- Cambiamos el tipo de datos para usarlo

Operando con input del usuario

- Usamos `raw_input()` para obtener información del usuario
- Cambiamos el tipo de datos para usarlo

```
# -*- coding: utf-8 -*-
print u"¿Edad?"
edad = int(raw_input())
print u"- - - - - - - - - -"
print u"Edad: %d" %edad
mayorEdad = (edad >= 18)
print u"¿Mayor de edad?: %s" %mayorEdad
```

Más funciones incluidas

https://docs.python.org/2/library/functions.html