Introducción a Python

Dora Suárez, Juan F. Pérez

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

juanferna.perez@urosario.edu.co

Primer Semestre de 2018

Contenidos

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

Imprimiendo en Python

`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"
```

Imprimiendo tildes y otros símbolos

¿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`

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

Errores

Tipos de errores al programar

Tipos de errores al programar

Léxico: uso de símbolos no reconocidos (fuera del vocabulario de Python)

Tipos de errores al programar

Léxico: uso de símbolos no reconocidos (fuera del vocabulario de Python)

Sintaxis: Gramática o puntuación incorrecta

Tipos de errores al programar

Léxico: uso de símbolos no reconocidos (fuera del vocabulario de Python)

Sintaxis: Gramática o puntuación incorrecta

Ejecución: Instrucciones no pueden ejecutarse

Tipos de errores al programar

Léxico: uso de símbolos no reconocidos (fuera del vocabulario de Python)

Sintaxis: Gramática o puntuación incorrecta

Ejecución: Instrucciones no pueden ejecutarse

Intención o Propósito: Programa se ejecuta pero no realiza la tarea esperada

¿Cómo reacciona Python a los errores?

Léxico: al intentar leer, Python nos dice que no entiende las palabras

¿Cómo reacciona Python a los errores?

Léxico: al intentar leer, Python nos dice que no entiende las palabras

Sintaxis: al intentar leer, Python nos dice que entiende las palabras pero no entiende la gramática o puntuación usadas

¿Cómo reacciona Python a los errores?

Léxico: al intentar leer, Python nos dice que no entiende las palabras

Sintaxis: al intentar leer, Python nos dice que entiende las palabras pero no entiende la gramática o puntuación usadas

Ejecución: Python entiende las palabras, la gramática y puntuación, pero al ejecutar el programa nos avisa que ha encontrado un error y termina sin culminar la tarea

¿Cómo reacciona Python a los errores?

Léxico: al intentar leer, Python nos dice que no entiende las palabras

Sintaxis: al intentar leer, Python nos dice que entiende las palabras pero no entiende la gramática o puntuación usadas

Ejecución: Python entiende las palabras, la gramática y puntuación, pero al ejecutar el programa nos avisa que ha encontrado un error y termina sin culminar la tarea

Intención/Propósito: Python entiende las palabras, la gramática y puntuación, y ejecuta sin errores, pero la tarea no es realizada satisfactoriamente (Python no lo sabe)

Errores: bugs

Llamamos *bugs* a los errores

Al proceso de buscar y resolver errores lo llamamos *debugging*

Funciones: Definiendo nuevas instrucciones

Funciones

No queremos tener todo el código en un solo bloque

Queremos definir en un solo sitio acciones repetitivas: *funciones*

Y llamamos estas funciones/instrucciones cada vez que las necesitemos

Funciones 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

Funciones en Python

Todas las funciones se definen al principio

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

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

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()
```

Algunos comentarios sobre las funciones

Cada nueva función es como una entrada en un diccionario para Python.

Todas las funciones nuevas las definimos al principio, antes del bloque de ejecución principal.

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)
```

¿Qué logramos con las funciones?

Definimos funciones que realizan tareas claramente demarcadas y con nombres naturales en lenguaje humano. Simplifica el desarrollo del programa.

¿Qué logramos con las funciones?

Definimos funciones que realizan tareas claramente demarcadas y con nombres naturales en lenguaje humano. Simplifica el desarrollo del programa.

Agrupamos sucesiones de instrucciones comunes. Simplifica el desarrollo y la lectura del programa.

¿Qué logramos con las funciones?

Definimos funciones que realizan tareas claramente demarcadas y con nombres naturales en lenguaje humano. Simplifica el desarrollo del programa.

Agrupamos sucesiones de instrucciones comunes. Simplifica el desarrollo y la lectura del programa.

Evitamos repetir las mismas instrucciones una y otra vez.

¿Qué logramos con las funciones?

Definimos funciones que realizan tareas claramente demarcadas y con nombres naturales en lenguaje humano. Simplifica el desarrollo del programa.

Agrupamos sucesiones de instrucciones comunes. Simplifica el desarrollo y la lectura del programa.

Evitamos repetir las mismas instrucciones una y otra vez.

Al definir nuevas instrucciones

Identifiquemos grupos de instrucciones que se usan repetidamente en el programa.

Más funciones incluidas

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

Resolviendo problemas

Resolviendo problemass

Pasos en la resolución de un problema:

- Definir el problema
- Planear la solución
- Implementar el plan
- Analizar la solución

Recomendaciones

Descomponer el problema en sub-problemas más sencillos

Recomendaciones

Descomponer el problema en sub-problemas más sencillos

Divide y vencerás (Divide and conquer)

Recomendaciones

Descomponer el problema en sub-problemas más sencillos

Divide y vencerás (Divide and conquer)

Implementar una solución a un problema más pequeño es más sencillo

...

Recomendaciones

Descomponer el problema en sub-problemas más sencillos

Divide y vencerás (Divide and conquer)

Implementar una solución a un problema más pequeño es más sencillo

...

También es más fácil verificar que la solución resuelva el problema mas pequeño...

Recomendaciones

Descomponer el problema en sub-problemas más sencillos

Divide y vencerás (Divide and conquer)

Implementar una solución a un problema más pequeño es más sencillo

...

También es más fácil verificar que la solución resuelva el problema mas pequeño...

Y corregir la solución de ser necesario

Y más recomendaciones

Definir funciones es útil para descomponer el problema

Y más recomendaciones

Definir funciones es útil para descomponer el problema

Cada función debe tener un **objetivo claro** y el número de líneas debe ser limitado para ...

Y más recomendaciones

Definir funciones es útil para descomponer el problema

Cada función debe tener un **objetivo claro** y el número de líneas debe ser limitado para ...

Entender qué hace la instrucción

Y más recomendaciones

Definir funciones es útil para descomponer el problema

Cada función debe tener un **objetivo claro** y el número de líneas debe ser limitado para ...

Entender qué hace la instrucción
Verificar su funcionamiento.

Y más recomendaciones

Definir funciones es útil para descomponer el problema

Cada función debe tener un **objetivo claro** y el número de líneas debe ser limitado para ...

Entender qué hace la instrucción
Verificar su funcionamiento.
Corregirla y **extender**la cuando sea necesario.

Y más recomendaciones

Definir funciones es útil para descomponer el problema

Cada función debe tener un **objetivo claro** y el número de líneas debe ser limitado para ...

Entender qué hace la instrucción
Verificar su funcionamiento.
Corregirla y **extender**la cuando sea necesario.

Las funciones también deben tener **nombres claros**, que faciliten entender qué hacen.