Introducción a Python Parte II

Matemática y Tecnología II
Maestría en Matemática

Prof. Robert Muñoz

Escuela de Matemática,
Universidad Autónoma de Santo Domingo (UASD)

Junio, 2024

Tabla de Contenido

print:
Es una función que nos permite mostrar en pantalla (imprimir) lo que le pasemos como parámetro.

```
1 print("hola mundo")
```

print e input

print:
Es una función que nos permite mostrar en pantalla (imprimir) lo que le pasemos como parámetro.

```
1 print("hola mundo")
```

input:
Es la función con la cual introducimos datos al programa usando el teclado. Todo lo introducido usando 'input()' se toma como un string.

```
1 input("Escribe un numero")
```

Tipos de Datos o Variables

- Numéricos:
  - Enteros (int)
  - Decimales (float)

Tipos de Datos o Variables

- Numéricos:
  - Enteros (int)
  - Decimales (float)
- Cadena de Caracteres:
  - Texto o palabras
  - Signos
  - Caracteres

Tipos de Datos o Variables

- Numéricos:
  - Enteros (int)
  - Decimales (float)
- Cadena de Caracteres:
  - Texto o palabras
  - Signos
  - Caracteres
- Booleanos:
  - Verdad (True)
  - Falso (False)

Tipos de Datos o Variables

- Numéricos:
  - Enteros (int)
  - Decimales (float)
- Cadena de Caracteres:
  - Texto o palabras
  - Signos
  - Caracteres
- Booleanos:
  - Verdad (True)
  - Falso (False)
- Secuenciales:
  - Lista (list)
  - Tupla (tuple)
  - Rango (range)

Tipos de Datos o Variables

- Numéricos:
  - Enteros (int)
  - Decimales (float)
- Cadena de Caracteres:
  - Texto o palabras
  - Signos
  - Caracteres
- Booleanos:
  - Verdad (True)
  - Falso (False)
- Secuenciales:
  - Lista (list)
  - Tupla (tuple)
  - Rango (range)
- Diccionarios

Numéricos

```
1 # asignacion de variables
2 x = 3
3 x = 4.5
```

Numéricos

```
# asignacion de variables
x = 3
x = 4.5
```

Escritura dinámica en Python

```
x = 3
y = 3.0
x == y
> True
```

Conversiones

- ‘int()‘: transforma a entero.
- ‘float()‘: transforma a decimal.
- ‘bool()‘: transforma a booleano.
- ‘str()‘: transforma a string (texto).

Operadores

- Operadores aritméticos.

Operadores

- Operadores aritméticos.
- Operadores relacionales.

Operadores

- Operadores aritméticos.
- Operadores relacionales.
- Operadores de asignación.

Operadores

- Operadores aritméticos.
- Operadores relacionales.
- Operadores de asignación.
- Operadores lógicos.

Operadores Aritméticos

- $+$, Suma
- $-$, Resta
- $*$, Multiplicación
- $/$, División
- $//$, División entera
- $**$, Exponente
- $\%$, Módulo

Operadores Relacionales

- $<$, Menor que $\quad \leq$, Menor o igual que
- $>$, Mayor que $\quad \geq$, Mayor o igual que
- $==$, Igual que $\quad !=$, Diferente que

Operadores de Asignación

- $=$, Asignación $\quad *=$, Asigna Producto
- $+=$, Asigna Suma $\quad /=$, Asigna División
- $-=$, Asigna Resta $\quad \%=$, Asigna Módulo

Operadores Lógicos

Operador Lógico	Sintaxis en Python
Y	and
O	or

Strings o Cadenas de Texto

Los objetos tipo string se escriben con comillas simples o dobles.

```
1 x = 'hola'
2 x = "hola"
```

Python discrimina entre mayúsculas y minúsculas.

Algunos métodos de string

Strings o Cadenas de Texto

Los objetos tipo string se escriben con comillas simples o dobles.

```
1 x = 'hola'
2 x = "hola"
```

Python discrimina entre mayúsculas y minúsculas.

Algunos métodos de string

- 'str.capitalize()': Retorna copia con primer caracter en mayúscula.
- 'str.lower()': Retorna copia con caracteres en minúsculas.
- 'str.upper()': Retorna copia con caracteres en mayúsculas.
- Para otros consulte: w3schools

Strings: Ejemplos

Capitalización

```
word = "Hello"
word.lower()
> 'hello'
```

Concatenar

```
"1" + "2"
> '12'
"Hola " + "amigo"
> "Hola amigo"
```

<2-> Repetición

```
"12" * 2
> "1212"
```

Indexing y Slicing

```
word = "Hello"
word[1]
> 'e'
```

```
word = "Hello"
word[1:3]
> 'el'
```

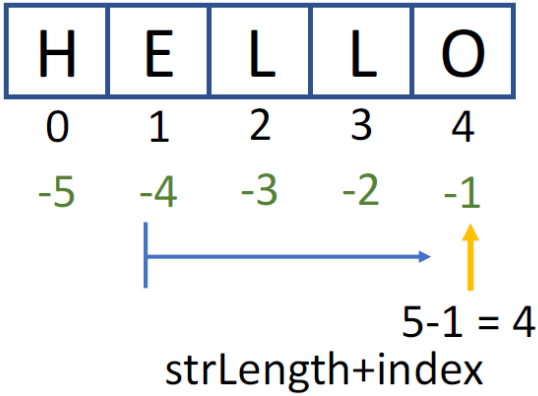
```
word = "Hello"
word[-4:-1]
> 'ell'
```

Indexing y Slicing

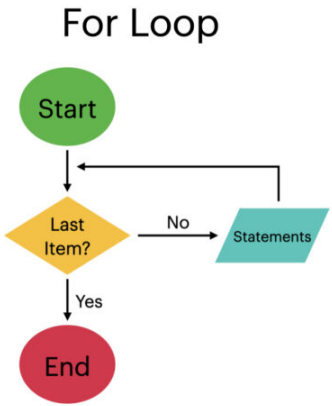
```
word = "Hello"
word[1]
> 'e'
```

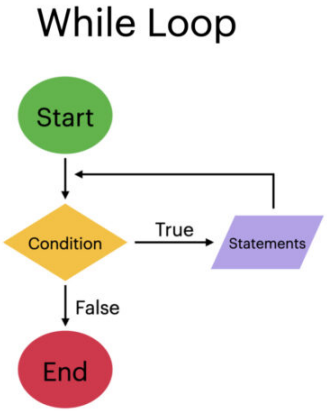
```
word = "Hello"
word[1:3]
> 'el'
```

```
word = "Hello"
word[-4:-1]
> 'ell'
```



Loops For y While





Loop For

Se utiliza para iterar sobre una secuencia (lista, tupla, string, etc.).

range

range(start, stop, step)
Retorna valores entre el ‘start‘ y el ‘stop-1‘, incrementando por ‘step‘.

Loop For

Se utiliza para iterar sobre una secuencia (lista, tupla, string, etc.).

range

range(start, stop, step)
Retorna valores entre el 'start' y el 'stop-1', incrementando por 'step'.

Sintaxis: Se usan dos puntos (:) y sangría (indentation).

```
1 for i in range(0, 10):
2     print(i)
```

Loop while

Ejecuta un bloque de instrucciones mientras una condición sea verdadera.

Loop while

Ejecuta un bloque de instrucciones mientras una condición sea verdadera. **Sintaxis:** Se usan dos puntos (:) y sangría.

```
1 i = 2
2 while i < 12:
3     print(i)
4     i += 3
```

Resultado:

Loop while

Ejecuta un bloque de instrucciones mientras una condición sea verdadera. **Sintaxis:** Se usan dos puntos (:) y sangría.

```
i = 2
while i < 12:
    print(i)
    i += 3
```

Resultado:

```
> 2
> 5
> 8
> 11
```

Condicional if

Se utiliza para tomar decisiones en el flujo del programa.

Estructura básica:

```
if statement:
    instructions
elif statement:
    instructions
else:
    instructions
```

Condicional if

Se utiliza para tomar decisiones en el flujo del programa.

Estructura básica:

```
if statement:
    instructions
elif statement:
    instructions
else:
    instructions
```

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Condicional if: Ejemplo

Ejemplo: Imprimir solo números pares

```
1 for i in range(0, 10):
2     if i % 2 == 0:
3         print(i)
```

Resultado:

```
1 > 0
2 > 2
3 > 4
4 > 6
5 > 8
```

Funciones

Bloque de código reutilizable que se ejecuta al ser llamado. Puede recibir parámetros y retornar un valor.

Funciones

Bloque de código reutilizable que se ejecuta al ser llamado. Puede recibir parámetros y retornar un valor.

Sintaxis:

```
def nombre_funcion(param1, param2):
    # bloque de codigo
    return valor
```

Funciones

Bloque de código reutilizable que se ejecuta al ser llamado. Puede recibir parámetros y retornar un valor.

Sintaxis:

```
def nombre_funcion(param1, param2):
    # bloque de codigo
    return valor
```

Ejemplo:

```
def multiplicar(x, y, z):
    return x * y * z

resultado = multiplicar(2, 3, 4)
# resultado es 24
```

Funciones

Bloque de código reutilizable que se ejecuta al ser llamado. Puede recibir parámetros y retornar un valor.

Sintaxis:

```
def nombre_funcion(param1, param2):
    # bloque de codigo
    return valor
```

Ejemplo:

```
def multiplicar(x, y, z):
    return x * y * z

resultado = multiplicar(2, 3, 4)
# resultado es 24
```

Nota: No usar nombres de funciones ya existentes en Python.

Funciones: Otro ejemplo

```
1 def valor_abs(val):
2     if val < 0:
3         return -val
4     return val
5
6 print(valor_abs(-7))
7 > 7
```

Función Anónima Lambda

Función pequeña y anónima. Puede tener varios argumentos pero solo una expresión.

Función Anónima Lambda

Función pequeña y anónima. Puede tener varios argumentos pero solo una expresión.

Ejemplo 1:

```
suma = lambda a, b: a + b
resultado = suma(3, 5)
print(resultado)
> 8
```

Función Anónima Lambda

Función pequeña y anónima. Puede tener varios argumentos pero solo una expresión.

Ejemplo 1:

```
suma = lambda a, b: a + b
resultado = suma(3, 5)
print(resultado)
> 8
```

Ejemplo 2:

```
operacion = lambda x: x * 2 + 3
print(operacion(4))
> 11
```

Listas

Las listas se definen con ‘[]‘ y son mutables (modificables).

```
lista1 = [11, 22, 33]
lista1
> [11, 22, 33]
```

Listas

Las listas se definen con '[]' y son mutables (modificables).

```
lista1 = [11, 22, 33]
lista1
> [11, 22, 33]
```

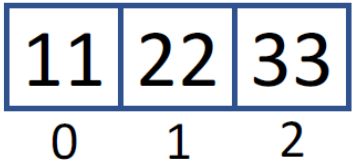
Indexing

```
lista1[1]
> 22
```

Listas

Las listas se definen con '[]' y son mutables (modificables).

```
lista1 = [11, 22, 33]
lista1
> [11, 22, 33]
```



Indexing

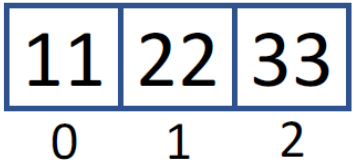
```
lista1[1]
> 22
```

¿Qué pasará con...

Listas

Las listas se definen con '[]' y son mutables (modificables).

```
lista1 = [11, 22, 33]
lista1
> [11, 22, 33]
```



Indexing

```
lista1[1]
> 22
```

¿Qué pasará con...

```
1 lista1[3]
2 > IndexError
```

Iterando en una lista

Iterar sobre elementos:

```
lista = [11, 22, 33]
for i in lista:
    print(i)
> 11
> 22
> 33
```

Iterando en una lista

Iterar sobre elementos:

```
lista = [11, 22, 33]
for i in lista:
    print(i)
> 11
> 22
> 33
```

Iterar sobre índices:

```
for i in range(len(lista)):
    print(lista[i])
```

Iterando en una lista

Iterar sobre elementos:

```
lista = [11, 22, 33]
for i in lista:
    print(i)
> 11
> 22
> 33
```

Iterar sobre índices:

```
for i in range(len(lista)):
    print(lista[i])
```

Las listas son mutables

```
1 lista = [11, 22, 33]
2 lista[1] = 95
3 print(lista)
4 > [11, 95, 33]
```

Manipulando listas (Parte 1)

Agregando elementos (‘append‘)

```
lista = [11, 22, 33]
lista.append(44)
lista
> [11, 22, 33, 44]
```

Manipulando listas (Parte 1)

Agregando elementos (‘append‘)

```
lista = [11, 22, 33]
lista.append(44)
lista
> [11, 22, 33, 44]
```

Eliminando por índice (‘pop‘)

```
1 lista = [11, 22, 33, 44]
2 valor_eliminado = lista.pop(2)
3 # valor_eliminado es 33
4 lista
5 > [11, 22, 44]
```

Manipulando listas (Parte 2)

Removiendo por valor (‘remove‘)

```
lista = [11, 22, 33, 44]
lista.remove(33)
lista
> [11, 22, 44]
```

Manipulando listas (Parte 2)

Removiendo por valor (‘remove‘)

```
lista = [11, 22, 33, 44]
lista.remove(33)
lista
> [11, 22, 44]
```

Extendiendo una lista (‘extend‘)

```
1 list1 = [1, 2, 3]
2 list2 = [4, 5, 6]
3 list1.extend(list2)
4 list1
5 > [1, 2, 3, 4, 5, 6]
```

List Comprehension

Manera concisa de crear listas a partir de otras secuencias.

List Comprehension

Manera concisa de crear listas a partir de otras secuencias.

Crear una lista de números

```
1 numeros = [x for x in range(5)]
2 print(numeros)
3 > [0, 1, 2, 3, 4]
```

List Comprehension

Manera concisa de crear listas a partir de otras secuencias.

Crear una lista de números

```
1 numeros = [x for x in range(5)]
2 print(numeros)
3 > [0, 1, 2, 3, 4]
```

Filtrar para obtener solo los pares

```
1 pares = [x for x in range(10) if x % 2 == 0]
2 print(pares)
3 > [0, 2, 4, 6, 8]
```

Tuplas

Las tuplas se definen con ‘()‘ y son **inmutables** (no se pueden cambiar).

```
tupla1 = ('honda', 'civic', 4, 2017)
for i in tupla1:
    print(i)
> honda
> civic
> 4
> 2017

# Esto daria error:
# tupla1[0] = 'toyota'
```

Diccionarios

Colección de pares 'clave:valor'. No están ordenados y son mutables.

Key	Value
'A12367'	'David Wu'
'A27691'	'Maria Sanchez'
'A16947'	'Tim Williams'
'A21934'	'Sarah Jones'

Sintaxis:

```
dict1 = {"A12367": "David Wu",
         "A27691": "Tim Williams",
         "A21934": "Sarah Jones"}
```

Diccionarios

Colección de pares 'clave:valor'. No están ordenados y son mutables.

Key	Value
'A12367'	'David Wu'
'A27691'	'Maria Sanchez'
'A16947'	'Tim Williams'
'A21934'	'Sarah Jones'

Accediendo a elementos:

```
1 dict1["A12367"]
2 > "David Wu"
```

Sintaxis:

```
dict1 = {"A12367": "David Wu",
         "A27691": "Tim Williams",
         "A21934": "Sarah Jones"}
```

Diccionarios

Colección de pares 'clave:valor'. No están ordenados y son mutables.

Key	Value
'A12367'	'David Wu'
'A27691'	'Maria Sanchez'
'A16947'	'Tim Williams'
'A21934'	'Sarah Jones'

Sintaxis:

```
dict1 = {"A12367": "David Wu",
         "A27691": "Tim Williams",
         "A21934": "Sarah Jones"}
```

Accediendo a elementos:

```
1 dict1["A12367"]
2 > "David Wu"
```

Agregando elementos:

```
1 dict1["B34567"] = "Robert"
```

Conjuntos

- No están ordenados y no permiten elementos duplicados.
- Soportan operaciones matemáticas como unión, intersección, etc.

Ejemplo de definición y operaciones

```
# Definicion de conjuntos
conjunto_a = {1, 2, 3, 4}
conjunto_b = {3, 4, 5, 6}

# Union (elementos de ambos, sin repetir)
union_ab = conjunto_a.union(conjunto_b)
print(union_ab)
> {1, 2, 3, 4, 5, 6}

# Interseccion (elementos en comun)
interseccion_ab = conjunto_a.intersection(conjunto_b)
print(interseccion_ab)
> {3, 4}
```