Manual de Python Básico

David Camilo Cortes Salazar

Análisis y Visualización de Datos

Talento Tech

Bogotá D.C.

2024

Manual de Python Básico

David Camilo Cortes Salazar

Docente: Carlos Palacios

Análisis y Visualización de Datos

Talento Tech

Bogotá D.C.

2024

Objetivo

El objetivo de este manual es proporcionar los conocimientos fundamentales necesarios para empezar a programar en Python.

Contenidos

Introducción a Python

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general. Fue creado por Guido van Rossum y lanzado por primera vez en 1991. Python es conocido por su sintaxis simple y fácil de leer, lo que lo convierte en una excelente opción tanto para principiantes como para programadores experimentados.

Aplicaciones Comunes

Python se utiliza en una variedad de campos y para múltiples propósitos, entre ellos:

- **Desarrollo Web**: El desarrollo de aplicaciones web con frameworks como Django y Flask facilitan la creacion de sistemas robustos y escalables.
- **Ciencia de Datos y Machine Learning**: Python cuenta con bibliotecas para análisis de datos y machine learning como los son Pandas, NumPy, y scikit-learn.
- **Automatización y Scripting**: Al ser un lenguaje interpretado con una sintaxis simple, Python es usado para automatizar tareas y/o crear robotizaciones.
- **Desarrollo de Software**: Python puede ser usado para construir software de escritorio.

Documentación

La documentación oficial de Python contiene explicaciones sobre cómo funciona todo Python y cómo hacer cosas con él. Es una herramienta de guía siempre disponible para ayudarnos a resolver problemas y entender cómo hacer las cosas correctamente.

Enlace de la documentación para python 3.x: https://docs.python.org/3/

Tipos de Datos

Python es un lenguaje de tipado dinámico, esto quiere decir que no necesitas declarar el tipo de variable cuando estas se crean.

Tipos Numéricos

1. **Enteros (`int`)**:
    - Números enteros.

```
x = 10
y = -5
```

2. **Flotantes (`float`)**:
    - Números con parte decimal.

```
pi = 3.14
e = 2.71
```

3. **Complejos (`complex`)**:
    - Números con parte real e imaginaria.

```
z = 1 + 2j
```

Cadenas de Texto (`str`)

Las cadenas son secuencias de caracteres, encerradas entre comillas (simples o dobles).

```
saludo = "Hola, mundo!"
nombre = 'David'
```

Booleanos (`bool`)

Los booleanos solo pueden tener dos valores: `True` o `False`.

```
verdadero = True
falso = False
```

Conversiones de Tipo

Se puede convertir entre tipos de datos usando funciones de conversión.

```
# Convertir entero a flotante
x = 10
y = float(x)  # y será 10.0

# Convertir flotante a entero
pi = 3.14
integer_pi = int(pi)  # integer_pi será 3

# Convertir entero a cadena
num = 100
num_str = str(num)  # num_str será "100"
```

Operadores Aritméticos

Los operadores aritméticos se utilizan para realizar operaciones de matemáticas básicas.

- Suma (+), Resta (-), Multiplicación (*), División (/)
- Potencia (**), División Entera (//), Módulo (%)

```
x = 10
y = 5
print(x + y)  # Suma: 15
print(x - y)  # Resta: 5
print(x * y)  # Multiplicación: 50
print(x / y)  # División: 2.0
print(x ** y) # Potencia: 100000
```

Operadores de Asignación

Los operadores de asignación se utilizan para asignar valores a variables.

- Asignación simple (=), Asignación con suma (+=), Asignación con resta (-=), etc.

```
x = 10
x += 5  # Equivalente a x = x + 5
print(x)  # 15
```

Operadores de Comparación

Los operadores de comparación se utilizan para comparar valores.

- Igualdad (==), No igual (!=), Mayor que (>), Menor que (<)

- Mayor o igual que (>=), Menor o igual que (<=)

```
x = 10
y = 5
print(x == y)  # False
print(x != y)  # True
print(x > y)   # True
```

Operadores Lógicos

Los operadores lógicos se utilizan para combinar expresiones lógicas.

- Y lógico (and), O lógico (or), No lógico (not)

```
x = 10
y = 5
z = 15
print(x > y and x < z)  # True
print(x > y or x < z)   # True
print(not(x > y))       # False
```

Operadores de Pertenencia y de Identidad

Los operadores de pertenencia y de identidad se utilizan para verificar si un valor pertenece a una secuencia y para comparar identidades de objetos, respectivamente.

- Pertenencia: `in`, `not in`
- Identidad: `is`, `is not`

```
lista = [1, 2, 3, 4, 5]
print(3 in lista)   # True
print(6 not in lista)  # True

a = 10
b = 10
print(a is b)   # True
```

Estructuras de Datos

Python tiene diferentes estructuras de datos que facilitan el uso y manipulación de datos.

Listas (`list`)

Las listas son colecciones ordenadas y mutables que se pueden modificar después de su creación.

- **Creación de listas**:

```
numeros = [1, 2, 3, 4, 5]
animales = ["perro", "gato", "loro"]
```

- **Acceso a elementos**:

```
primer_animal = animal[0]  # "perro"
```

- **Modificación de elementos**:

```
animales[1] = "tigre"  # ["tigre", "gato", "loro"]
```

- **Agregar elementos**:

```
animales.append("leon")  # ["tigre", "gato", "loro", "leon"]
```

- **Eliminar elementos**:

```
animales.remove("tigre")  # ["gato", "loro", "leon"]
```

- **Slicing**:

```
corte_numeros = numeros[1:4]  # [2, 3, 4]
```

Tuplas (`tuple`)

Las tuplas son colecciones ordenadas e inmutables que no se pueden modificar después de su creación.

- **Creación de tuplas**:

```
coordenadas = (10.0, 20.0)
colores = ("rojo", "verde", "azul")
```

- **Acceso a elementos**:

```
primer_color = colors[0]  # "rojo"
```

- **Slicing**:

```
corte_colores = colors[1:3]  # ("verde", "azul")
```

Conjuntos (`set`)

Los conjuntos son colecciones desordenadas de elementos únicos que no permiten duplicados. Como lo seria un conjunto en matematicas.

- **Creación de conjuntos**:

```
numeros = {1, 2, 3, 4, 5}
animales = {"perro", "gato", "loro"}
```

- **Agregar elementos**:

```
numeros.add(6)  # {1, 2, 3, 4, 5, 6}
```

- **Eliminar elementos**:

```
animales.remove("gato")  # {"perro", "loro"}
```

- **Operaciones de conjuntos**:

```
set_a = {1, 2, 3}
set_b = {3, 4, 5}

union = set_a | set_b  # {1, 2, 3, 4, 5}
interseccion = set_a & set_b  # {3}
diferencia = set_a - set_b  # {1, 2}
```

Diccionarios (`dict`)

Los diccionarios son colecciones desordenadas de pares clave-valor. Cada clave debe ser única. Como lo seria un hashmap

- **Creación de diccionarios**:

```
persona = {
    "nombre": "David",
    "edad": 23,
    "ciudad": "Bogotá"
}
```

- **Acceso a valores**:

```
nombre = persona["nombre"]  # "Alice"
```

- **Modificación de valores**:

```
persona["edad"] = 30  # {"nombre": "David", "edad": 30, "ciudad": Bogotá}"
```

- **Agregar pares clave-valor**:

```
persona["email"] = "david@example.com"  # {"nombre": "David", "edad": 30,
"ciudad": Bogotá, "email": "david@example.com}"
```

- **Eliminar pares clave-valor**:

```
del persona["ciudad"]  #  {"nombre": "David", "edad": 30, "email":
"david@example.com}"
```

- **Métodos útiles**:

```
keys = persona.keys()  # dict_keys(['nombre', 'edad', 'email'])
values = persona.values()  # dict_values(['David', 30, 'david@example.com'])
items = person.items()  # dict_items([('nombre', 'David'), ('edad', 30),
('email', 'david@example.com')])
```

Funciones Incluidas en el Lenguaje

Python incluye una serie de funciones integradas que son parte del lenguaje.

Función `print()`

La función `print()` se utiliza para imprimir mensajes en la consola.

```
print("¡Hola, mundo!") # Imprime "Hola, mundo!"
```

También se puede imprimir el contenido de variables y expresiones.

```
nombre = "David"
edad = 23
print("Nombre:", nombre, "-", "Edad:", edad) # Imprime "Nombre: David - Edad: 23"
```

Puedes utilizar el operador `+` para concatenar cadenas y combinarlas en un solo mensaje que se imprimirá.

```
nombre = "David"
edad = 23
print("Nombre: " + nombre + ", Edad: " + str(edad)) # Imprime "Nombre: David,
Edad: 23
```

hay que tener en cuenta que si se combinan diferentes tipos de datos, como cadenas y números, se tienen que convertir los números a cadenas con la función `str()`.

Uso de f-strings

Las f-strings son una característica poderosa de Python que te permite insertar valores de variables en cadenas de texto.

```
nombre = "David"
edad = 23
print(f"Nombre: {nombre}, Edad: {edad}")  # Imprime "Nombre: David, Edad: 23
```

Con f-strings, se inserta el nombre de la variable entre llaves `{}` dentro de la cadena de texto.

Función `input()`

La función `input()` se utiliza para obtener la entrada del usuario desde la consola.

```
nombre = input("Introduce tu nombre: ")
print("Hola,", nombre)
```

Función `len()`

La función `len()` se utiliza para obtener la longitud de estructuras como cadenas, listas, tuplas, etc.

```
cadena = "Hola, mundo!"
print("Longitud:", len(cadena)) # Imprime "Longitud: 12"

lista = [1, 2, 3, 4, 5]
print("Longitud:", len(lista)) # Imprime "Longitud: 5"
```

Función `range()`

La función `range()` se utiliza para generar una secuencia de números.

```
for i in range(5):
    print(i)  # Imprime los números del 0 al 4

for i in range(1, 6):
    print(i)  # Imprime los números del 1 al 5
```

Función `type()`

La función `type()` se utiliza para obtener el tipo de un objeto.

```
x = 5
print(type(x))  # Imprime <class 'int'>

y = "Hola, mundo!"
print(type(y))  # Imprime <class 'str'>
```

Función `help()`

La función `help()` se utiliza para obtener ayuda sobre el uso de objetos, funciones y módulos.

```
help(print)  # Muestra la documentación de la función print
```

Declaraciones y Control de Flujo

Python tiene diferentes estructuras para manejar el flujo de ejecución del programa.

Declaración if

La declaración `if` se utiliza para ejecutar un bloque de código si una condición es verdadera.

```
x = 10
if x > 5:
    print("x es mayor que 5")
```

Declaración if-else

La declaración `if-else` se utiliza para ejecutar un bloque de código si una condición es verdadera y otro bloque si la condición es falsa.

```
x = 10
if x > 5:
    print("x es mayor que 5")
else:
    print("x es menor o igual que 5")
```

Declaración if-elif-else

La declaración `if-elif-else` se utiliza para comprobar múltiples condiciones.

```
x = 10
if x > 10:
    print("x es mayor que 10")
elif x == 10:
    print("x es igual a 10")
else:
    print("x es menor que 10")
```

Bucles for

El bucle `for` se utiliza para iterar sobre una secuencia (como una lista, tupla, diccionario, conjunto o cadena).

```
aniamles = ["perro", "gato", "loro"]
for animal in animales:
    print(animal)
```

El bucle `for` también puede usarse con la función `range` para iterar sobre una secuencia de números.

```
for i in range(5):
    print(i)  # Imprime números de 0 a 4
```

Bucles while

El bucle `while` se utiliza para repetir un bloque de código mientras una condición sea verdadera.

```
x = 0
while x < 5:
    print(x)
    x += 1
```

Declaraciones break y continue

- **break**: Termina el bucle inmediatamente.

```
for i in range(10):
    if i == 5:
        break
    print(i)  # Imprime números de 0 a 4
```

- **continue**: Omite el codigo restante en la iteración actual y pasa a la siguiente iteración.

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)  # Imprime números impares del 1 al 9
```

Bucles anidados

Los bucles pueden anidarse, es decir, puedes usar un bucle dentro de otro bucle.

```
for i in range(3):
    for j in range(2):
        print(f"i = {i}, j = {j}")
```

Comprensión de listas

La comprensión de listas ofrece una forma concisa de crear listas. Es útil para aplicar una expresión a cada elemento de una secuencia.

```
cuadrados = [x**2 for x in range(10)]
```

Sentencias pass

La declaración `pass` es una operación nula.

```
def funcion_nula():
    pass
```

Funciones y Módulos

Las funciones y módulos son componentes que permiten organizar y reutilizar el código.

Funciones

Las funciones son bloques de código que realizan una tarea específica y pueden ser reutilizadas.

- **Definición de Funciones**:

```
def saludo(name):
    return f"Hola, {name}!"

mensaje = saludo("David")
print(saludo)  # Imprime "Hola, David!"
```

- **Parámetros y Argumentos**:
  - Los parámetros son variables en la definición de la función.
  - Los argumentos son los valores pasados a la función.

```
def suma(a, b): # Parametros: a, b
    return a + b

resultado = suma(3, 5) # Argumentos: 3, 5
print(resultado)  # Imprime 8
```

- **Valores Predeterminados de Parámetros**:

```
def saludo(nombre, frase="Hola"):
    return f"{frase}, {nombre}!"

mensaje = saludo("David")
print(mensaje)  # Imprime "Hola, David!"

mensaje = saludo("David", "Buenos días")
print(mensaje)  # Imprime "Buenos días, David!"
```

- **Argumentos y Parámetros con Nombre**:

```
def describir_persona(nombre, edad, ciudad):
    return f"{nombre} tiene {edad} años y vive en {ciudad}."

descripcion = describir_persona(age=23, name="David", city="Bogotá")
print(descripcion)  # Imprime "David tiene 23 años y vive en Bogotá"
```

- **Funciones Lambda**: Las funciones lambda son funciones anónimas y pequeñas que se definen en una sola línea.

```
suma = lambda a, b: a + b
print(suma(3, 5))  # Imprime 8
```

Módulos

Los módulos son archivos que contienen definiciones y sentencias que ayudan a organizar el código y pueden ser reutilizados en diferentes programas.

- **Creación y Uso de Módulos**:
  - Crea un archivo llamado `mimodulo.py` que contenga lo siguiente::

```
def saludo(nombre):
    return f"Hola, {nombre}!"
```

  - En otro archivo, puedes importar y usar el módulo de esta manera:

```
import mimodulo

mensaje = mimodulo.saludo("David")
print(mensaje)  # Imprime "Hola, David!"
```

- **Importar Funciones Específicas**:

```
from mimodulo import saludo

mensaje = saludo("David")
print(mensaje)  # Imprime "Hola, David!"
```

- **Importar con Alias**:

```
import mimodulo as mm

mensaje = mm.saludo("David")
print(mensaje)  # Imprime "Hola, David!"
```

- **Uso del Directorio de Módulos**:
  - los módulosse pueden organizar en directorios mediante una estrictura como esta:

```
mi_paquete/
    __init__.py
    mimodulo.py
```

  - `__init__.py` puede estar vacío o contener código de inicialización del paquete.

Uso de Módulos Estándar

Python viene con una biblioteca estándar que incluye muchos módulos útiles.

- **Módulo math**:

  Esta herramienta inclye diferentes tipos de constantes y funciones matematicas.

  Para hacer uso del modulo se debe importar de la siguiente manera:

```
import math
```

  Veamos algunas de las funciones que incluye:

  - Raiz Cuadrada:

    Esta funcion recibe un número como argumento y retorna su raiz cuadrada.

    Ejemplo: $\sqrt{(16)} = 4$

```
raiz = math.sqrt(16) # raiz cuadrada

print(raiz)  # Imprime 4.0
```

  - Potencia:

    Esta funcion recibe dos n'umeros como argumentos y retorna la potencia de estos números. El primero sera la base y el segundo el exponente.

Ejemplo: $4^2 = 16$

```
pot = math.pow(4, 2) # potencia

print(pot)  # Imprime 16
```

- Logaritmo:

  Esta funcion recibe dos números como argumentos y retorna el logaritmo de estos números. El primero sera el argumento y el segundo el base.

  Ejemplo: $log_4(16) = 2$

```
log = math.log(16, 4) # logaritmo

print(log)  # Imprime 2
```

- Función Techo:

  Esta funcion recibe un número como argumento y retorna el mayor número entero mas cercano.

  Ejemplo: $\lceil 4.2 \rceil = 5$

```
techo = math.ceil(4.2) # Techo

print(techo)  # Imprime 5
```

- Función Piso:

  Esta funcion recibe un número como argumento y retorna el menor número entero mas cercano.

  Ejemplo: $\lfloor 4.8 \rfloor = 4$

```
piso = math.floor(4.8) # Piso

print(piso)  # Imprime 4
```

- Factorial:

  Esta funcion recibe un número como argumento y retorna su factorial.

  Ejemplo: $5! = 120$

```
fac = math.factorial(5) # Factorial

print(fac)  # Imprime 120
```

- Maximo Común Divisor:

  Esta funcion recibe dos números como argumentos y retorna el maximo común divisor entre ellos.

  Ejemplo: $\gcd(25, 120) = 5$

```
gcd = math.gcd(5,120) # Maximo común divisor

print(gcd)  # Imprime 5
```

- Valor Absoluto:

  Esta funcion recibe un número como argumento y retorna su valor absoluto.

  Ejemplo: $|-10| = 10$

```
vabs = math.fabs(-10) # Valor absoluto

print(vasb)  # Imprime 10
```

- Función Exponencial:

  Esta funcion recibe un número como argumento y retorna el exponencial de dicho número.

  Ejemplo: $e^3 = 20.085536923187668$

```
fexp = math.exp(3) # Exponencial

print(fexp)  # Imprime 20.085536923187668
```

- **Módulo datetime**:

  Esta herramienta proporciona una variedad de clases para representar, manipular, formatear y utilizar fechas y horas en diferentes formatos.

  El siguiente ejemplo muestra como acceder a la fecha y hora actuales:

```
from datetime import datetime

now = datetime.now() # Fecha y hora actual
print(now)  # Imprime la fecha y hora actuales
```

Bibliografía

- Python Documentation. Documentación Python
- Real Python. Real Python
- Programiz Python Tutorials. Programiz Python Tutorials
- Python Tutoriales. Python Tutoriales