

Instituto de formación técnica superior N° 18

Diagramación Lógica - Apuntes

Curso: 1° Año

Profesor: Bonini Juan Ignacio

Ciclo lectivo: 2023

Carga horaria: 5 hs cátedra/semana

Régimen: Cuatrimestral

Historia del lenguaje de programación Python


Python fue creado a fines de la década de 1980 por Guido van Rossum, un programador holandés. Van Rossum desarrolló Python como un proyecto personal mientras trabajaba en el Centro de Matemáticas y Ciencias de la Computación (CWI) en los Países Bajos. Su objetivo era crear un lenguaje de programación fácil de leer, con una sintaxis clara y legible.

La primera versión pública de Python, la versión 0.9.0, fue lanzada en febrero de 1991. A lo largo de los años, Python fue ganando popularidad gracias a sus características distintivas, como su enfoque en la legibilidad del código y su énfasis en la productividad y la simplicidad.

En 2000, se lanzó la versión 2.0 de Python, que introdujo mejoras significativas en el lenguaje. Python 2 se convirtió en la versión dominante durante muchos años y se utilizó ampliamente en la comunidad de desarrollo.

Sin embargo, en 2008, Guido van Rossum anunció el desarrollo de Python 3, una versión con importantes cambios y mejoras en comparación con Python 2. Python 3 introdujo modificaciones en la sintaxis y la biblioteca estándar para abordar deficiencias y ambigüedades en la versión anterior. Aunque hubo alguna resistencia inicial debido a la incompatibilidad entre Python 2 y Python 3, con el tiempo, la comunidad adoptó Python 3 como la versión principal.

Python ha sido ampliamente utilizado en diversos campos y aplicaciones, incluyendo desarrollo web, análisis de datos, inteligencia artificial, aprendizaje automático, automatización de tareas, desarrollo de videojuegos y más. Su popularidad ha sido impulsada por una gran cantidad de



bibliotecas y marcos de trabajo de código abierto que brindan funcionalidades adicionales y facilitan el desarrollo de aplicaciones.

En la actualidad, Python es uno de los lenguajes de programación más populares y está respaldado por una activa comunidad de desarrolladores que contribuyen con mejoras, bibliotecas y recursos educativos. Su facilidad de uso, legibilidad y versatilidad han contribuido a su éxito y crecimiento continuo a lo largo de los años.

¿Por qué empezamos a aprender a programar con Python?

Hay varias razones por las cuales Python es una excelente opción para comenzar a aprender programación:


Sintaxis clara y legible: Python tiene una sintaxis simple y fácil de entender. Utiliza indentación en lugar de llaves o paréntesis para delimitar bloques de código, lo que lo hace más legible y menos propenso a errores de formato. Esto facilita la comprensión del código, especialmente para principiantes.

Fácil de aprender: Python se considera uno de los lenguajes de programación más amigables para principiantes. Su sintaxis simple y su enfoque en la legibilidad hacen que el proceso de aprendizaje sea más suave y menos intimidante. Además, cuenta con una amplia comunidad de desarrolladores que brindan recursos y apoyo adicional.

Versatilidad: Python es un lenguaje versátil que se utiliza en una amplia gama de campos y aplicaciones. Puedes usarlo para desarrollar aplicaciones web, análisis de datos, aprendizaje automático, automatización de tareas, creación de scripts y mucho más. Aprender Python te brinda habilidades que son aplicables en diversos entornos y te permite explorar diferentes áreas de interés.

Amplia comunidad y recursos: Python cuenta con una comunidad de desarrolladores muy activa y solidaria. Hay numerosos recursos educativos disponibles, como tutoriales, cursos en línea, documentación y foros de discusión. Además, Python tiene una gran cantidad de bibliotecas y marcos de trabajo que simplifican el desarrollo y te permiten aprovechar funcionalidades adicionales sin tener que escribir todo desde cero.

Oportunidades profesionales: Python es ampliamente utilizado en la industria y hay una alta demanda de profesionales con habilidades en Python. Aprender Python te brinda oportunidades en el campo laboral, ya sea que busques un empleo como programador, científico de datos o ingeniero de aprendizaje automático.



En resumen, Python es una excelente opción para comenzar a aprender programación debido a su sintaxis clara, facilidad de aprendizaje, versatilidad, comunidad activa y oportunidades profesionales. Es un lenguaje que te brinda una base sólida para desarrollar habilidades de programación y explorar diferentes áreas de interés en el mundo de la tecnología.

¿Qué tipo de lenguaje de programación es python?

Python es un lenguaje de programación interpretado y de sintaxis fuertemente tipada.

Python es conocido por ser un lenguaje de programación de alto nivel y de sintaxis clara. En este sentido, es fuertemente tipado, lo que significa que se requiere la declaración y el manejo explícito de los tipos de datos. Cada variable en Python tiene un tipo asociado y no se permite mezclar tipos de manera implícita en las operaciones. Por ejemplo, no se puede sumar una cadena de texto y un número sin convertirlos explícitamente.


Sin embargo, Python también proporciona características de tipado dinámico. Esto significa que no es necesario declarar el tipo de una variable antes de usarla, y el tipo de una variable puede cambiar durante la ejecución del programa.

En resumen, Python es un lenguaje interpretado con una sintaxis clara y legible. Es fuertemente tipado, lo que requiere el manejo explícito de los tipos de datos, pero también permite el tipado dinámico.

Python es interpretado pero compilado, ¿Cómo es la cosa?

Python es comúnmente conocido como un lenguaje de programación interpretado, lo que significa que el código fuente de Python se ejecuta directamente por un intérprete de Python línea por línea. Sin embargo, la implementación de Python más común, conocida como CPython (escrita en C), utiliza un enfoque híbrido de compilación e interpretación.

En el proceso de ejecución de un programa de Python, el código fuente se compila en un código de bytes llamado "código de byte de Python" o "archivo .pyc". Este archivo .pyc contiene instrucciones en un formato más eficiente para la ejecución por parte del intérprete de Python. Esta compilación se realiza una vez, a menos que el archivo .pyc se elimine o se realicen cambios en el código fuente.



Cuando se ejecuta el programa posteriormente, el intérprete de Python lee y ejecuta el código compilado (.pyc) en lugar del código fuente original (.py), lo que proporciona una ejecución más rápida que la interpretación directa del código fuente.

Es importante destacar que esta compilación a bytecode no genera código de máquina específico de la plataforma como lo haría un compilador tradicional. En cambio, el bytecode es interpretado por el intérprete de Python para generar los resultados de la ejecución.

En resumen, Python utiliza un proceso híbrido de compilación e interpretación donde el código fuente se compila en bytecode y luego se interpreta. Esto proporciona una ejecución más rápida en comparación con una interpretación directa, aunque no genera código de máquina específico de la plataforma como lo hacen los lenguajes completamente compilados.

Sintaxis básica de python

La sintaxis básica de Python es relativamente sencilla y fácil de entender. Algunos conceptos fundamentales:

Variables y asignación

Puedes almacenar valores en variables utilizando el operador de asignación (=). Python es un lenguaje de tipado dinámico, por lo que no es necesario declarar explícitamente el tipo de variable.

Por ejemplo:

```
mensaje = "Hola, Python"
numero = 42
```

Comentarios

Los comentarios son líneas de código que no se ejecutan y se utilizan para agregar explicaciones o notas al código. En Python, los comentarios comienzan con el símbolo #.

Por ejemplo:

```
# Esto es un comentario
```

Sintaxis básica de python - Tipos de datos

Python tiene varios tipos de datos, tanto básicos como complejos, que te permiten almacenar y manipular diferentes tipos de información.

Tipos de datos básicos

- Números enteros (int): Representan números enteros, como 1, 2, -5, etc.
- Números de punto flotante (float): Representan números decimales, como 3.14, 2.5, -0.75, etc.
- Cadenas de texto (str): Representan una secuencia de caracteres entre comillas, como "Hola", 'Python', "42", etc.
- Booleanos (bool): Representan valores verdaderos o falsos (True o False).

Tipos de datos complejos

- Listas (list): Representan una colección ordenada y modificable de elementos. Se pueden almacenar diferentes tipos de datos en una lista. Las listas se encierran entre corchetes []. Por ejemplo, [1, 2, 3], ["a", "b", "c"], [1, "hello", True], etc.
- Tuplas (tuple): Son similares a las listas, pero son inmutables, lo que significa que no se pueden modificar una vez creadas. Las tuplas se encierran entre paréntesis (). Por ejemplo, (1, 2, 3), ("a", "b", "c"), (1, "hello", True), etc.
- Diccionarios (dict): Representan una colección de pares clave-valor. Cada elemento del diccionario consiste en una clave única y su correspondiente valor. Los diccionarios se encierran entre llaves {}. Por ejemplo, {"nombre": "Juan", "edad": 25, "ciudad": "Madrid"}, etc.
- Conjuntos (set): Representan una colección desordenada de elementos únicos. Los conjuntos no permiten elementos duplicados y se utilizan para realizar operaciones de conjuntos, como intersecciones, uniones, etc. Los conjuntos se encierran entre llaves {} o se crean con la función set(). Por ejemplo, {1, 2, 3}, {"a", "b", "c"}, set([1, 2, 3]), etc.

Estructuras de control

- Condicionales (if, else, elif): Permiten tomar decisiones basadas en condiciones.
- Bucles (for, while): Permiten repetir un bloque de código varias veces.

Sintaxis básica de python - Tipos de datos

Ejemplos de tipos de datos básicos:

```
# Números enteros (int)
edad = 25
saldo_bancario = -1000
numero_de_cuenta = 1234567890

pi = 3.14159
altura = 1.85
precio = 9.99

# Cadenas de texto (str)
nombre = "Juan"
mensaje = 'Hola, ¿cómo estás?'
codigo_postal = "12345"

# Booleanos (bool)
es_mayor_de_edad = True
tiene_licencia = False
esta_activo = True
```

Ejemplo de declaración de variables con todos los tipos de datos complejos:

```
# Listas (list)
numeros = [1, 2, 3, 4, 5]
nombres = ["Juan", "María", "Pedro"]
mezclado = [1, "Hola", True, 3.14]

# Tuplas (tuple)
coordenadas = (10, 20)
meses = ("Enero", "Febrero", "Marzo")
punto = (3.5, -2.7)

# Diccionarios (dict)
persona = {"nombre": "Juan", "edad": 25, "ciudad": "Madrid"}
pelicula = {"titulo": "El Padrino", "año": 1972, "director": "Francis Ford Coppola"}

# Conjuntos (set)
numeros_primos = {2, 3, 5, 7, 11}
vocales = {"a", "e", "i", "o", "u"}
```

Sintaxis básica de python - Nombres de variables

En Python, existen algunas reglas y convenciones que debes seguir al nombrar variables.

Algunos ejemplos de nombres de variables incorrectos:

Los nombres de variables no pueden comenzar con un número. Debe comenzar con una letra o un guión bajo _.

Los nombres de variables no pueden contener espacios. En su lugar, puedes usar guiones bajos _.

Los nombres de variables no pueden contener caracteres especiales como el signo de dólar (\$), el signo de exclamación (!), entre otros. Solo se permiten letras, números y guiones bajos.

No puedes utilizar palabras reservadas del lenguaje Python como nombres de variables. Estas palabras tienen un significado especial en Python y se utilizan para funciones y estructuras de control. Algunas palabras reservadas comunes incluyen for, if, while, def, class, entre otras.

Recuerda que los nombres de variables en Python son sensibles a mayúsculas y minúsculas, lo que significa que "nombre" y "Nombre" se consideran nombres de variables diferentes. Además, es recomendable seguir las convenciones de estilo de Python, como utilizar letras minúsculas y guiones bajos para separar palabras en nombres de variables (por ejemplo, nombre_completo, precio_unitario). Esto ayuda a que tu código sea más legible para otros programadores.

Sintaxis básica de python - Trabajando con listas

El manejo de listas en Python es una parte fundamental de la programación. Las listas son estructuras de datos que te permiten almacenar y manipular colecciones ordenadas de elementos. Algunos conceptos básicos sobre el manejo de listas en Python:

Crear una lista

Puedes crear una lista utilizando corchetes [] y separando los elementos por comas. Los elementos de una lista pueden ser de diferentes tipos.

Por ejemplo

```
lista_numeros = [1, 2, 3, 4, 5]
lista_nombres = ["Juan", "María", "Pedro"]
lista_mixta = [1, "Hola", True, 3.14]
```

Acceder a los elementos de una lista

Puedes acceder a los elementos de una lista utilizando índices. Los índices en Python comienzan desde 0 para el primer elemento.

Por ejemplo

```
lista = ["manzana", "banana", "cereza"]
print(lista[0]) # Muestra "manzana"
print(lista[2]) # Muestra "cereza"
```

Modificar elementos de una lista

Puedes modificar elementos individuales de una lista asignándoles un nuevo valor utilizando el índice.

Por ejemplo

```
lista = ["manzana", "banana", "cereza"]
lista[1] = "kiwi"
print(lista) # Muestra ["manzana", "kiwi", "cereza"]
```

Obtener la longitud de una lista

Puedes usar la función `len()` para obtener la cantidad de elementos en una lista.

Por ejemplo

```
lista = [1, 2, 3, 4, 5]
longitud = len(lista)
print(longitud) # Muestra 5
```

Agregar elementos a una lista

Puedes agregar elementos a una lista utilizando el método `append()`.

Por ejemplo

```
lista = [1, 2, 3]
lista.append(4)
print(lista) # Muestra [1, 2, 3, 4]
```


Eliminar elementos de una lista

Puedes eliminar elementos de una lista utilizando el método `remove()` pasando el valor del elemento que deseas eliminar.

Por ejemplo

```
lista = [1, 2, 3, 4, 5]
lista.remove(3)
print(lista)  # Muestra [1, 2, 4, 5]
```

Sintaxis básica de python - Trabajando con diccionarios

El manejo de diccionarios en Python es muy útil para almacenar y acceder a datos utilizando una estructura de clave-valor. Algunos conceptos básicos sobre el manejo de diccionarios en Python:

Crear un diccionario

Puedes crear un diccionario utilizando llaves `{}` y separando las claves y valores por dos puntos `:`.

Por ejemplo

```
diccionario = {"clave1": valor1, "clave2": valor2, "clave3": valor3}
```

Acceder a los valores de un diccionario

Puedes acceder a los valores de un diccionario utilizando las claves correspondientes.

Por ejemplo

```
diccionario = {"nombre": "Juan", "edad": 25, "ciudad": "Madrid"}
print(diccionario["nombre"])  # Muestra "Juan"
print(diccionario["edad"])    # Muestra 25
```

Modificar valores de un diccionario

Puedes modificar los valores de un diccionario asignando un nuevo valor a una clave existente.

Por ejemplo

```
diccionario = {"nombre": "Juan", "edad": 25, "ciudad": "Madrid"}
diccionario["edad"] = 30
print(diccionario) # Muestra {"nombre": "Juan", "edad": 30, "ciudad": "Madrid"}
```

Agregar elementos a un diccionario

Puedes agregar nuevos elementos a un diccionario asignando un valor a una nueva clave.

Por ejemplo

```
diccionario = {"nombre": "Juan", "edad": 25}
diccionario["ciudad"] = "Madrid"
print(diccionario) # Muestra {"nombre": "Juan", "edad": 25, "ciudad": "Madrid"}
```

Eliminar elementos de un diccionario

Puedes eliminar elementos de un diccionario utilizando la instrucción del y la clave correspondiente.

Por ejemplo

```
diccionario = {"nombre": "Juan", "edad": 25, "ciudad": "Madrid"}
del diccionario["edad"]
print(diccionario) # Muestra {"nombre": "Juan", "ciudad": "Madrid"}
```


Verificar si una clave existe en un diccionario

Puedes utilizar el operador in para verificar si una clave existe en un diccionario.

Por ejemplo

```
diccionario = {"nombre": "Juan", "edad": 25, "ciudad": "Madrid"}
if "nombre" in diccionario:
    print("La clave 'nombre' existe en el diccionario")
```

Sintaxis básica de python - For



El bucle for es una estructura de control en Python que permite iterar sobre una secuencia de elementos, como una lista, una tupla, una cadena de texto o incluso un diccionario. El bucle for se utiliza para realizar una serie de instrucciones o acciones repetidamente para cada elemento de la secuencia.

La sintaxis básica del bucle for en Python es la siguiente:

```
for elemento in secuencia:  
    # Bloque de código a ejecutar para cada elemento
```

Donde:

elemento es una variable que tomará el valor de cada elemento de la secuencia en cada iteración.

secuencia es la secuencia de elementos sobre la cual se va a iterar.

Dentro del bucle for, puedes realizar cualquier acción que desees con el elemento actual. Esto puede incluir operaciones, cálculos, impresiones en pantalla, modificaciones en variables, llamadas a funciones, entre otros.

Ejemplos

Iterar sobre una lista

```
frutas = ["manzana", "plátano", "naranja"]  
for fruta in frutas:  
    print(fruta)
```

Iterar sobre una cadena de texto

```
mensaje = "Hola, mundo!"  
for caracter in mensaje:  
    print(caracter)
```

Iterar sobre un rango de números

```
for numero in range(1, 6):  
    print(numero)
```

Recuerde que el **range** recibe distintos valores, en este caso recibe en el primer parámetro el número desde el cual se comienza a iterar y en el segundo parámetro el número hasta el cual NO queremos llegar.

Iterar sobre un diccionario


```
estudiantes = {"Juan": 18, "María": 20, "Pedro": 19}  
for nombre, edad in estudiantes.items():  
    print(nombre, "tiene", edad, "años")
```


En cada iteración del bucle for, la variable elemento toma el valor del siguiente elemento de la secuencia y se ejecuta el bloque de código asociado. El bucle continúa hasta que se han procesado todos los elementos de la secuencia.

El bucle for es una herramienta poderosa que te permite automatizar tareas repetitivas y procesar una secuencia de elementos de manera eficiente en Python.

Ejercicios

1. Ejercicios de variables básicas

- 
- 1.1. Declara una variable entera y muéstrala en pantalla
 - 1.2. Declara una variable de punto flotante y muéstrala en pantalla
 - 1.3. Declara una cadena de texto y muéstrala en pantalla
 - 1.4. Declara una variable booleana y muéstrala en pantalla
 - 1.5. Declara una lista de números y muéstrala en pantalla
 - 1.6. Declara una tupla de nombres y muéstrala en pantalla
 - 1.7. Declara un diccionario de edades y muéstralo en pantalla
 - 1.8. Declara un conjunto de letras y muéstralo en pantalla
 2. Ejercicios de variables con operaciones básicas
 - 2.1. Declara una variable con el resultado de una operación matemática y muéstrala en pantalla
 - 2.2. Declara una variable con una concatenación de cadenas de texto y muéstrala en pantalla
 - 2.3. Declara una variable con una cadena de texto multilínea y muéstrala en pantalla
 - 2.4. Declara una variable booleana y muestra su negación en pantalla
 - 2.5. Declara una lista de números y muestra un elemento específico en pantalla
 - 2.6. Declara una cadena de texto y muestra su longitud en pantalla
 - 2.7. Declara un diccionario de edades y muestra el valor asociado a una clave específica en pantalla
 - 2.8. Declara una variable con un número de punto flotante y muestra solo su parte entera en pantalla
 - 2.9. Declara una lista vacía, agrega elementos y muéstrala en pantalla
 - 2.10. Declara una cadena de texto y muestra solo los primeros tres caracteres en pantalla
 3. Ejercicios básicos con listas
 - 3.1. Crea una lista de números y muestra la suma de todos los elementos
 - 3.2. Crea una lista de números y muestra la suma de los números pares en la lista.
 - 3.3. Crea una lista de números y muestra la suma de los números impares en la lista.
 - 3.4. Crea una lista de números y muestra la suma de los números mayores que 10 en la lista.
 4. Ejercicios básicos con diccionarios
 - 4.1. Crea un diccionario con información de una persona (nombre, edad, ciudad) y muestra todos los datos en pantalla.
 - 4.2. Crea una lista de diccionarios que contengan información de varias personas y muestra el nombre de cada persona en pantalla.
 - 4.3. Crea una lista de diccionarios que contengan información de libros (título, autor, año de publicación) y muestra el título y el autor de cada libro en pantalla.

- 
- 4.4. Crea un diccionario con nombres de frutas y sus cantidades correspondientes.
Muestra solo las frutas que tienen una cantidad mayor que 10.