

Codificación de Algoritmos con Python

Este documento introduce los conceptos fundamentales para la codificación de algoritmos utilizando el lenguaje de programación Python. Se abordan las plataformas de desarrollo, el proceso de elaboración de código desde una perspectiva de lenguaje interpretado y los pilares de la programación estructurada y orientada a objetos, incluyendo variables, tipos de datos, estructuras de control (condicionales y bucles), y la definición de clases y objetos.

Índice

1	Codificación de Algoritmos	2
1.1	Plataformas de Programación	2
1.2	Elaboración de Código	2
1.2.1	Lenguajes Interpretados vs. Compilados	2
1.2.2	Sintaxis Básica en Python	2
1.3	Programación Estructurada	3
1.3.1	Variables y Tipos de Datos	3
1.3.2	Asignación y Operadores Aritméticos	3
1.3.3	Entrada y Salida de Datos	3
1.3.4	Estructuras de Control: Selección	4
1.3.5	Estructuras de Control: Iteración	4
1.3.6	Ejemplo Práctico Integrador	4
1.4	Programación Orientada a Objetos (POO)	4
1.4.1	Conceptos Fundamentales	4
1.4.2	Atributos y Métodos	5
2	Recursos Adicionales y Cursos Gratuitos	5

1 Codificación de Algoritmos

Una vez que se ha diseñado un algoritmo, el siguiente paso lógico es la **codificación**, que consiste en traducir dicho algoritmo a un lenguaje que una computadora pueda entender y ejecutar. A este producto se le conoce como **programa** o **código fuente**. Para este material, utilizaremos Python, un lenguaje moderno, legible y potente, ideal para principiantes.

1.1 Plataformas de Programación

Una **plataforma de programación** es un entorno de software que permite a los desarrolladores escribir, probar y ejecutar código. Para Python, las opciones más comunes son:

- **Entornos de Desarrollo Integrado (IDEs):** Son aplicaciones completas que incluyen un editor de texto, herramientas de depuración y una consola integrada. Son ideales para proyectos grandes. Ejemplos populares son **Visual Studio Code**, **PyCharm** y **Spyder**.
- **Notebooks Interactivos:** Permiten combinar código ejecutable, texto y gráficos. Son excelentes para el análisis de datos y la enseñanza, ya que permiten ejecutar el código por bloques. **Jupyter Notebook** y **Google Colab** son los más utilizados.
- **Editor de Texto y Terminal:** El enfoque más simple. Ideal para scripts rápidos y para entender el proceso de ejecución sin ayudas adicionales. Consiste en escribir código en un editor (como Sublime Text o Notepad++) y ejecutarlo desde la terminal.

1.2 Elaboración de Código

El proceso de dar vida a un algoritmo en Python implica entender cómo el lenguaje interpreta y ejecuta nuestras instrucciones.

1.2.1 Lenguajes Interpretados vs. Compilados

A diferencia de lenguajes como FORTRAN o C++, que son **compilados**, Python es un lenguaje **interpretado** [5].

Imagina la diferencia como un traductor de libros vs. un intérprete en una conferencia:

- **Compilador (Traductor):** Lee el libro completo (código fuente) y lo traduce a otro idioma (lenguaje máquina), generando un nuevo libro (archivo ejecutable). Si encuentra un error, se detiene y no produce nada.
- **Intérprete (Intérprete en vivo):** Escucha una frase (línea de código), la traduce y la dice inmediatamente. Si no entiende una frase, se detiene en ese preciso instante.

El proceso en Python es el siguiente:

1. **Escribir el código:** Se guarda el programa en un archivo con extensión `.py`.
2. **Ejecutar con el intérprete:** Se abre una terminal y se invoca al intérprete de Python, pasándole el nombre del archivo.

```
python mi_programa.py
```

Ejemplo 1: Ejecución de un script de Python en la terminal.

1.2.2 Sintaxis Básica en Python

- **Comentarios:** Cualquier texto que sigue al símbolo `#` es ignorado.
- **Sensibilidad a Mayúsculas:** Python distingue entre mayúsculas y minúsculas. `edad` y `Edad` serían dos variables distintas.
- **Indentación:** Python utiliza la indentación (4 espacios) para definir bloques de código. Un error en la indentación es uno de los fallos más comunes para un principiante.

```

1 # Este código producirá un error
2 edad = 20
3 if edad >= 18:
4     print("Eres mayor de edad.") # ERROR: Falta
      indentación

```

Ejemplo 2: Error de indentación (*IndentationError*).

1.3 Programación Estructurada

La **programación estructurada** es un paradigma que busca mejorar la claridad y calidad del código mediante el uso de tres estructuras de control básicas: secuencia, selección (condicionales) e iteración (bucles) [4].

1.3.1 Variables y Tipos de Datos

Las variables son etiquetas para guardar datos. Python usa **tipado dinámico**: no se declara el tipo, se infiere del valor asignado.

Los tipos de datos primitivos más comunes son: `int`, `float`, `str` y `bool`. El tipo de una variable puede cambiar si se le reasigna un valor de otro tipo.

```

1 # La variable 'valor' cambia de tipo
2 valor = 100
3 print(f"Valor: {valor}, Tipo: {type(valor)}")
4
5 valor = "Cien" # Ahora 'valor' es de tipo
   str
6 print(f"Valor: {valor}, Tipo: {type(valor)}")

```

Ejemplo 3: Ejemplo de tipado dinámico.

1.3.2 Asignación y Operadores Aritméticos

El operador de asignación utilizado en Python es `=`. Los operadores aritméticos siguen un orden de precedencia estándar.

Para ejemplificar su uso, podemos hacer un programa que haga uso de los operadores en la fórmula del cálculo de área de un rectángulo.

Cuadro 1: Principales operadores aritméticos en Python.

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División (float)
//	División Entera
%	Módulo (residuo)
**	Potencia

```

1 # Algoritmo para calcular el área de un
   rectángulo
2 print("--- Calculadora de Área ---")
3 str_base = input("Introduce la base (cm): ")
4 str_altura = input("Introduce la altura (cm)
   : ")
5
6 # Conversión de tipos
7 base = float(str_base)
8 altura = float(str_altura)
9 area = base * altura
10
11 # Salida de datos
12 print(f"El área del rectángulo es: {area} cm
   ^2")

```

Ejemplo 4: Cálculo del área de un rectángulo.

1.3.3 Entrada y Salida de Datos

- **Salida:** `print()` muestra datos en pantalla.
- **Entrada:** `input()` solicita datos al usuario y siempre los devuelve como texto (`str`).

Las **f-strings** (`f`) son la forma moderna y recomendada de formatear cadenas de texto, permitiendo incrustar variables directamente.

```

1 nombre_usuario = input("Cual es tu nombre? ")
2
3 str_edad = input("Cual es tu edad? ")
4 edad_usuario = int(str_edad)
5 edad_futura = edad_usuario + 10
6
7 # La f antes de las comillas activa el
   formateo
8 print(f"Hola, {nombre_usuario}!")

```

```
8 print(f"En 10 años tendrás {edad_futura} años.")
```

Ejemplo 5: *Uso de f-string para formatear salida.*

1.3.4 Estructuras de Control: Selección

Las estructuras de selección permiten decidir qué conjunto de instrucciones ejecutar basándose en si una condición lógica es verdadera o falsa.

- **Sentencia if:** Evalúa una condición. Si es verdadera, ejecuta el bloque indentado.
- **Sentencia else:** Es opcional y ofrece un camino alternativo si la condición inicial no se cumple.
- **Sentencia elif:** Permite evaluar múltiples condiciones en cadena.

```
1 calificacion = 85
2
3 if calificacion >= 90:
4     print("Excelente")
5 elif calificacion >= 60:
6     print("Aprobado")
7 else:
8     print("Reprobado")
```

Ejemplo 6: *Uso de condicionales.*

1.3.5 Estructuras de Control: Iteración

La iteración permite repetir un bloque de instrucciones un número determinado de veces o mientras se cumpla una condición.

- **Bucle for:** Ideal para recorrer secuencias de datos (como listas o rangos numéricos) donde conocemos la cantidad de elementos.
- **Bucle while:** Repite el bloque de instrucciones *mientras* una condición lógica sea verdadera.

```
1 # Iteracion con for (imprime 0, 1, 2)
2 for i in range(3):
3     print(f"Iteracion {i}")
```

Ejemplo 7: *Uso de bucle for.*

1.3.6 Ejemplo Práctico Integrador

El siguiente ejemplo calcula el Índice de Masa Corporal (IMC), uniendo todos los conceptos vistos. La fórmula es: $IMC = \frac{peso}{altura^2}$.

```
1 # 1. Bienvenida
2 print("--- Calculadora de IMC ---")
3
4 # 2. Entrada de datos
5 str_peso = input("Introduce tu peso en kg: ")
6 str_altura = input("Introduce tu altura en m: ")
7
8 # 3. Conversion de tipos
9 peso = float(str_peso)
10 altura = float(str_altura)
11
12 # 4. Proceso (calculo del IMC)
13 imc = peso / (altura ** 2)
14
15 # 5. Salida de resultados
16 # :.2f redondea el resultado a dos decimales
17 print(f"Tu IMC es: {imc:.2f}")
```

Ejemplo 8: *Calculadora de Índice de Masa Corporal (IMC).*

1.4 Programación Orientada a Objetos (POO)

La Programación Orientada a Objetos es un paradigma que organiza el diseño de software alrededor de datos u **objetos**, en lugar de funciones y lógica [4].

1.4.1 Conceptos Fundamentales

- **Clase (class):** Es la plantilla o molde abstracto. Define qué atributos y métodos tendrán los objetos creados a partir de ella.
- **Objeto:** Es una instancia concreta de una clase. Es la entidad que existe en la memoria de la computadora.

1.4.2 Atributos y Métodos

Dentro de una clase, definimos:

- **Atributos:** Variables internas del objeto que almacenan su estado (ej. color, marca).
- **Métodos:** Funciones definidas dentro de la clase que determinan el comportamiento (ej. encender).

```
1 class Coche:
2     # Constructor: Inicializa los atributos
3     def __init__(self, marca, color):
4         self.marca = marca
5         self.color = color
6
7     # Metodo: Define un comportamiento
8     def encender(self):
9         print(f"El {self.marca} {self.color}
10             arranco.")
11
12 # Creacion de Objetos (Instanciacion)
13 mi_auto = Coche("Toyota", "Rojo")
14 mi_auto.encender()
```

Ejemplo 9: Clases y Objetos en Python.

2 Recursos Adicionales y Cursos Gratuitos

Para continuar aprendiendo Python, se recomiendan los siguientes recursos de acceso libre, reconocidos por su alta calidad:

- **El Tutorial Oficial de Python:** La fuente principal de información. Es claro, completo y siempre está actualizado a la última versión del lenguaje.
- **freeCodeCamp:** Ofrece un curso muy completo de Computación Científica con Python que cubre desde lo más básico hasta temas avanzados, con proyectos prácticos.
- **Python 3 Programming (Coursera):** Una especialización de la Universidad de Michigan, ideal para principiantes. Los videos y materiales son gratuitos (modo auditoría).
- **Canal de Corey Schafer en YouTube:** Considerado uno de los mejores canales para aprender Python. Sus tutoriales son detallados, bien explicados y cubren una gran variedad de temas.

Referencias

- [1] A. B. Downey, *Think Python: How to Think Like a Computer Scientist*. O'Reilly Media, 2015.
- [2] C. R. Severance, *Python for Everybody: Exploring Data in Python 3*. CreateSpace Independent Publishing Platform, 2016.
- [3] A. Sweigart, *Automate the Boring Stuff with Python*. No Starch Press, 2019.
- [4] L. Joyanes Aguilar, *Fundamentos de programación: algoritmos, estructura de datos y objetos*. McGraw-Hill, 2003.
- [5] F. J. Ceballos, *C/C++: curso de programación*. Ra-Ma, 2006.
- [6] Python Software Foundation. *Python 3.12.0 documentation*. Disponible en: <https://docs.python.org/3/>
- *Software y Plataformas* —
- [7] Python Software Foundation. *Python Logo*. Disponible en: <https://es.wikipedia.org/wiki/Python#/media/Archivo:Python-logo-notext.svg>
- [8] Microsoft Corporation. *Visual Studio Code Icon*. Disponible en: https://es.m.wikipedia.org/wiki/Archivo:Visual_Studio_Code_1.35_icon.svg
- [9] Project Jupyter. *Jupyter Logo*. Disponible en: https://commons.wikimedia.org/wiki/File:Jupyter_logo.svg
- [10] Sublime HQ Pty Ltd. *Sublime Text*. Disponible en: <https://www.sublimetext.com>