Contents

1 Variables en Python **3**
1.1 Declaración de variables . . . 3
1.2 Nombres de variables . . . 3
1.3 Asignación múltiple . . . 3
1.4 Asignación múltiple con el mismo valor . . . 3
1.4.1 Variables globales y locales . . . 3
1.4.2 Variables globales . . . 3
1.5 Variables locales . . . 3
1.6 Dirección de memoria . . . 4

2 Tipos de datos **4**
2.1 Tipos de datos numéricos . . . 4
2.1.1 Enteros . . . 4
2.1.2 Flotantes . . . 4
2.1.3 Complejos . . . 4
2.2 Tipos de datos secuenciales . . . 4
2.2.1 Cadenas . . . 5
2.2.2 Listas . . . 5
2.2.3 Tuplas . . . 5
2.3 Manejo de cadenas . . . 5
2.3.1 Longitud de una cadena . . . 5
2.3.2 Índices de una cadena . . . 5
2.3.3 Subcadenas . . . 5
2.3.4 Cadenas con formato . . . 5
2.4 Tipos booleanos . . . 6
2.5 Procesar entrada del usuario . . . 6
2.5.1 Entrada del usuario . . . 6
2.5.2 Conversión de la entrada del usuario . . . 6

3 Entrada y salida de datos **6**
3.1 Imprimir en la consola . . . 6
3.2 Formato de impresión . . . 6
3.3 Entrada del usuario . . . 6
3.4 Conversión de la entrada del usuario . . . 7

4 Operadores **7**
4.1 Operadores aritméticos . . . 7
4.2 Operadores de asignación . . . 7
4.3 Operadores de comparación . . . 8
4.4 Operadores lógicos . . . 8

5 Estructuras de control **8**
5.1 Estructura if . . . 8
5.2 Estructura else . . . 8
5.3 Estructura elif . . . 9
5.4 Estructura while . . . 9
5.5 Estructura for . . . 9
5.6 Break y Continue . . . 9

6 Funciones **9**
6.1 Definición de funciones . . . 10
6.2 Llamada a funciones . . . 10
6.3 Argumentos de funciones . . . 10
6.4 Parámetros de funciones . . . 10
6.5 Valores de retorno . . . 10

7 Clases y objetos **10**
7.1 Definición de clases . . . 10
7.2 Creación de objetos . . . 11
7.3 Métodos de objetos . . . 11
7.4 El método __init__() . . . 11
7.5 El método del objeto __del__() . . . 11

8 Módulos **12**
8.1 Importación de módulos . . . 12
8.2 Uso de módulos . . . 12
8.3 Alias de módulos . . . 12
8.4 Módulos integrados . . . 12

9 Métodos numéricos **12**
9.1 Algoritmo de Horner . . . 12
9.2 Algoritmo de Bisección . . . 13
9.3 Algoritmo de Newton-Raphson . . . 13

1 Variables en Python

Una variable es un espacio de memoria que se utiliza para almacenar un valor. En Python, las variables no necesitan ser declaradas con ningún tipo en particular y pueden incluso cambiar de tipo después de haber sido creadas.

1.1 Declaración de variables

Para declarar una variable en Python, se utiliza el operador de asignación `=`. La sintaxis es la siguiente:

```
variable = valor
```

Donde `variable` es el nombre de la variable y `valor` es el valor que se le asigna. Por ejemplo:

```
x = 5
y = "Hola"
```

1.2 Nombres de variables

Los nombres de las variables en pueden contener letras, números y guiones bajos (_). Sin embargo, el nombre de una variable no puede comenzar con un número. Además, Python distingue entre mayúsculas y minúsculas, por lo que las variables `variable`, `Variable` y `VARIABLE` son diferentes.

1.3 Asignación múltiple

Python permite asignar un valor a varias variables al mismo tiempo. Por ejemplo:

```
x, y, z = 5, 10, 15
```

1.4 Asignación múltiple con el mismo valor

Tambien nos permite asignar el mismo valor a varias variables al mismo tiempo. Por ejemplo:

```
x = y = z = 5
```

1.4.1 Variables globales y locales

Una variable que se declara fuera de una función es una variable global, mientras que una variable que se declara dentro de una función es una variable local. Una variable global se puede utilizar en cualquier parte del programa, mientras que una variable local solo se puede utilizar dentro de la función en la que se declara.

1.4.2 Variables globales

Para declarar una variable global dentro de una función, se utiliza la palabra clave `global`. Por ejemplo:

```
def myfunc():
  global x
  x = "fantastic"
```

1.5 Variables locales

Si se declara una variable dentro de una función, esta será local, a menos que se utilice la palabra clave `global`. Por ejemplo:

```
def myfunc():
  x = "fantastic"
```

1.6 Dirección de memoria

En Python, se puede obtener la dirección de memoria de una variable utilizando la función `id()`. Por ejemplo:

```
x = 5
print(id(x))
```

2 Tipos de datos

2.1 Tipos de datos numéricos

Se admiten los siguientes tipos de datos numéricos:

- `int` (enteros)
- `float` (flotantes)
- `complex` (complejos)

2.1.1 Enteros

Los enteros son números enteros, positivos o negativos, sin decimales, de longitud ilimitada. Por ejemplo:

```
x = 1
y = 35656222554887711
z = -3255522
```

2.1.2 Flotantes

Los flotantes son números reales, positivos o negativos, que contienen uno o más decimales. Por ejemplo:

```
x = 1.10
y = 1.0
z = -35.59
```

2.1.3 Complejos

Los números complejos se escriben con una `j` como parte imaginaria. Por ejemplo:

```
x = 3+5j
y = 5j
z = -5j
```

2.2 Tipos de datos secuenciales

Python admite los siguientes tipos de datos secuenciales:

- `str` (cadenas)
- `list` (listas)
- `tuple` (tuplas)

2.2.1 Cadenas

Las cadenas se definen entre comillas simples o dobles. Por ejemplo:

```
x = "Hola"
y = 'Hola'
```

2.2.2 Listas

Las listas se definen entre corchetes y pueden contener cualquier tipo de dato. Por ejemplo:

```
x = ["manzana", "platano", "cereza"]
y = [1, 5, 7, 9, 3]
z = [True, False, False]
```

2.2.3 Tuplas

Las tuplas se definen entre paréntesis y pueden contener cualquier tipo de dato. Por ejemplo:

```
x = ("manzana", "platano", "cereza")
y = (1, 5, 7, 9, 3)
z = (True, False, False)
```

2.3 Manejo de cadenas

2.3.1 Longitud de una cadena

Para obtener la longitud de una cadena, se utiliza la función `len()`. Por ejemplo:

```
a = "Hola, Mundo!"
print(len(a))
```

2.3.2 Índices de una cadena

Los índices de una cadena comienzan en 0. Por ejemplo:

```
a = "Hola, Mundo!"
print(a[1])
```

2.3.3 Subcadenas

Se pueden obtener subcadenas utilizando la notación de rebanadas. Por ejemplo:

```
b = "Hola, Mundo!"
print(b[2:5])
```

2.3.4 Cadenas con formato

Se pueden utilizar las cadenas con formato para insertar valores en una cadena. Por ejemplo:

```
edad = 36
txt = "Mi nombre es Juan, y tengo {}"
print(txt.format(edad))
```

2.4 Tipos booleanos

Python tiene un tipo de datos booleanos, que solo puede tener dos valores: `True` o `False`. Por ejemplo:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

2.5 Procesar entrada del usuario

2.5.1 Entrada del usuario

Para obtener la entrada del usuario, se utiliza la función `input()`. Por ejemplo:

```
print("Ingresa tu nombre:")
x = input()
print("Hola, " + x)
```

2.5.2 Conversión de la entrada del usuario

La entrada del usuario se almacena como una cadena. Para convertirla a un tipo de dato numérico, se utiliza la función `int()` o `float()`. Por ejemplo:

```
print("Ingresa un numero:")
x = int(input())
print("El numero es " + str(x))
```

3 Entrada y salida de datos

3.1 Imprimir en la consola

Para imprimir en la consola, se utiliza la función `print()`. Por ejemplo:

```
print("Hola, Mundo!")
```

3.2 Formato de impresión

Se pueden utilizar las cadenas con formato para insertar valores en una cadena. Por ejemplo:

```
edad = 36
txt = "Mi nombre es Juan, y tengo {}"
print(txt.format(edad))
```

3.3 Entrada del usuario

Para obtener la entrada del usuario, se utiliza la función `input()`. Por ejemplo:

```
print("Ingresa tu nombre:")
x = input()
print("Hola, " + x)
```

3.4 Conversión de la entrada del usuario

La entrada del usuario se almacena como una cadena. Para convertirla a un tipo de dato numérico, se utiliza la función `int()` o `float()`. Por ejemplo:

```
print("Ingresa un numero:")
x = int(input())
print("El numero es " + str(x))
```

4 Operadores

4.1 Operadores aritméticos

Los operadores aritméticos se utilizan para realizar operaciones matemáticas. Por ejemplo:

```
x = 5
y = 3
print(x + y)
print(x - y)
print(x * y)
print(x / y)
print(x % y)
print(x ** y)
print(x // y)
```

4.2 Operadores de asignación

Los operadores de asignación se utilizan para asignar valores a las variables. Por ejemplo:

```
x = 5
x += 3
x -= 3
x *= 3
x /= 3
x %= 3
x **= 3
x //= 3
```

Donde cada uno significa lo siguiente:

- `x += 3` es equivalente a `x = x + 3`
- `x -= 3` es equivalente a `x = x - 3`
- `x *= 3` es equivalente a `x = x * 3`
- `x /= 3` es equivalente a `x = x / 3`
- `x %= 3` es equivalente a `x = x % 3`
- `x **= 3` es equivalente a `x = x ** 3`
- `x //= 3` es equivalente a `x = x // 3`

4.3 Operadores de comparación

Los operadores de comparación se utilizan para comparar dos valores. Por ejemplo:

```
x = 5
y = 3
print(x == y)
print(x != y)
print(x > y)
print(x < y)
print(x >= y)
print(x <= y)
```

4.4 Operadores lógicos

Los operadores lógicos se utilizan para combinar declaraciones condicionales. Por ejemplo:

```
x = 5
print(x > 3 and x < 10)
print(x > 3 or x < 4)
print(not(x > 3 and x < 10))
```

5 Estructuras de control

Una estructura de control es un bloque de código que decide qué línea de código se ejecutará a continuación. Python admite las siguientes estructuras de control:

- `if` (si)
- `else` (si no)
- `elif` (si no, si)
- `while` (mientras)
- `for` (para)
- `break` (romper)
- `continue` (continuar)

5.1 Estructura if

La estructura `if` se utiliza para tomar decisiones basadas en condiciones. Por ejemplo:

```
a = 33
b = 200
if b > a:
  print("b es mayor que a")
```

5.2 Estructura else

La estructura `else` se utiliza para ejecutar un bloque de código si la condición es falsa. Por ejemplo:

```
a = 33
b = 33
if b > a:
  print("b es mayor que a")
else:
  print("b no es mayor que a")
```

5.3 Estructura elif

La estructura `elif` se utiliza para ejecutar un bloque de código si la condición anterior es falsa. Por ejemplo:

```
a = 33
b = 33
if b > a:
  print("b es mayor que a")
elif a == b:
  print("a y b son iguales")
```

5.4 Estructura while

La estructura `while` se utiliza para ejecutar un bloque de código mientras la condición sea verdadera. Por ejemplo:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

5.5 Estructura for

La estructura `for` se utiliza para iterar sobre una secuencia (como una lista, tupla, conjunto o diccionario). Por ejemplo:

```
frutas = ["manzana", "platano", "cereza"]
for x in frutas:
  print(x)
```

5.6 Break y Continue

La estructura `break` se utiliza para salir del bucle. Por ejemplo:

```
frutas = ["manzana", "platano", "cereza"]
for x in frutas:
  print(x)
  if x == "platano":
    break
```

La estructura `continue` se utiliza para saltar la iteración actual y continuar con la siguiente. Por ejemplo:

```
frutas = ["manzana", "platano", "cereza"]
for x in frutas:
  if x == "platano":
    continue
  print(x)
```

6 Funciones

Una función es un bloque de código que solo se ejecuta cuando se llama. Python admite funciones predefinidas, así como funciones definidas por el usuario.

6.1 Definición de funciones

En Python, una función se define utilizando la palabra clave `def`. Por ejemplo:

```
def my_function():
  print("Hola desde una funcion")
```

6.2 Llamada a funciones

Para llamar a una función, simplemente se escribe el nombre de la función seguido de paréntesis. Por ejemplo:

```
def my_function():
  print("Hola desde una funcion")

my_function()
```

6.3 Argumentos de funciones

Los argumentos son los valores que se pasan a la función. Por ejemplo:

```
def my_function(fname):
  print(fname + " Refsnes")

my_function("Emil")
my_function("Tobias")
my_function("Linus")
```

6.4 Parámetros de funciones

Los parámetros son los nombres de las variables que se utilizan al definir una función. Por ejemplo:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Emil", "Refsnes")
```

6.5 Valores de retorno

Para devolver un valor de una función, se utiliza la palabra clave `return`. Por ejemplo:

```
def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

7 Clases y objetos

7.1 Definición de clases

En Python, una clase se define utilizando la palabra clave `class`. Por ejemplo:

```
class MyClass:
  x = 5
```

7.2 Creación de objetos

Un objeto es una instancia de una clase. Para crear un objeto, se utiliza la misma sintaxis que para llamar a una función. Por ejemplo:

```
p1 = MyClass()
print(p1.x)
```

7.3 Métodos de objetos

Los métodos son funciones que pertenecen a un objeto. Por ejemplo:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hola, mi nombre es " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

7.4 El método __init__()

El método __init__() se utiliza para asignar valores a las propiedades del objeto cuando se crea el objeto. Por ejemplo:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

7.5 El método del objeto __del__()

El método __del__() se utiliza para destruir un objeto. Por ejemplo:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __del__(self):
    print("El objeto ha sido destruido")

p1 = Person("John", 36)
del p1
```

8 Módulos

8.1 Importación de módulos

Los módulos son archivos que contienen un conjunto de funciones que se pueden incluir en un programa. Para importar un módulo, se utiliza la palabra clave `import`. Por ejemplo:

```
import mymodule
```

8.2 Uso de módulos

Una vez que se ha importado un módulo, se pueden utilizar las funciones que contiene. Por ejemplo:

```
import mymodule

mymodule.greeting("Jonathan")
```

8.3 Alias de módulos

Se puede crear un alias cuando se importa un módulo, utilizando la palabra clave `as`. Por ejemplo:

```
import mymodule as mx

a = mx.person1["age"]
print(a)
```

8.4 Módulos integrados

Python tiene un conjunto de módulos integrados que se pueden utilizar sin necesidad de instalarlos. Por ejemplo:

```
import platform

x = platform.system()
print(x)
```

9 Métodos numéricos

9.1 Algoritmo de Horner

El algoritmo de Horner es un método para evaluar polinomios. Por ejemplo:

```
def horner(p, x):
    n = len(p)
    result = p[0]
    for i in range(1, n):
        result = result * x + p[i]
    return result
```

Este algoritmo evalúa un polinomio de la forma $p(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ en el punto x. Un ejemplo de uso de este algoritmo es el siguiente:

```
p = [1, 2, 3]
x = 2
print(horner(p, x))
```

Donde la lista p representa el polinomio $p(x) = 1 + 2x + 3x^2$ y $x = 2$.

9.2 Algoritmo de Bisección

El algoritmo de bisección es un método para encontrar raíces de una función. Por ejemplo:

```
def biseccion(f, a, b, tol):
    if f(a) * f(b) > 0:
        return None
    while (b - a) / 2.0 > tol:
        midpoint = (a + b) / 2.0
        if f(midpoint) == 0:
            return midpoint
        elif f(a) * f(midpoint) < 0:
            b = midpoint
        else:
            a = midpoint
    return (a + b) / 2.0
```

Este algoritmo encuentra una raíz de la función $f(x)$ en el intervalo $[a, b]$ con una tolerancia de tol. Un ejemplo de uso de este algoritmo es el siguiente:

```
def f(x):
    return x**2 - 2

a = 1
b = 2
tol = 1e-6
print(biseccion(f, a, b, tol))
```

Donde la función $f(x) = x^2 - 2$ tiene una raíz en el intervalo $[1, 2]$.

9.3 Algoritmo de Newton-Raphson

El algoritmo de Newton-Raphson es un método para encontrar raíces de una función. Por ejemplo:

```
def newton_raphson(f, df, x0, tol):
    while abs(f(x0)) > tol:
        x0 = x0 - f(x0) / df(x0)
    return x0
```

Este algoritmo encuentra una raíz de la función $f(x)$ con derivada $f'(x)$ y una tolerancia de tol. Un ejemplo de uso de este algoritmo es el siguiente:

```
def f(x):
    return x**2 - 2

def df(x):
    return 2 * x

x0 = 1
tol = 1e-6
print(newton_raphson(f, df, x0, tol))
```

Donde la función $f(x) = x^2 - 2$ tiene una raíz en el punto $x = 1$.