Aprendiendo Python 3

Intermedio

Ing. Jezrrel Valles

jezrrel.valles@ia.center

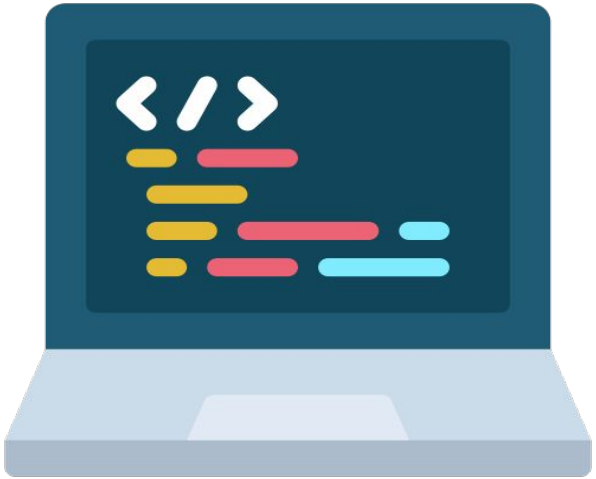
(656) 586-2120

Programación

Desarrolla habilidades

- Resolución de problemas
- Pensamiento algorítmico
- Creatividad
- Análisis
- Pensamiento crítico

Una vez que adquieras las habilidades necesarias, descubrirás el infinito potencial de tus conocimientos.



¿Nuevo en Python 3?

Python

Lenguaje de programación de alto nivel que se destaca por su legibilidad y facilidad de escritura tanto para humanos como para computadoras.

Funciona como un intérprete…

1. Lee el código
2. Analiza
3. Interpreta en tiempo real como una secuencia ordenada de instrucciones.

Python for Everybody

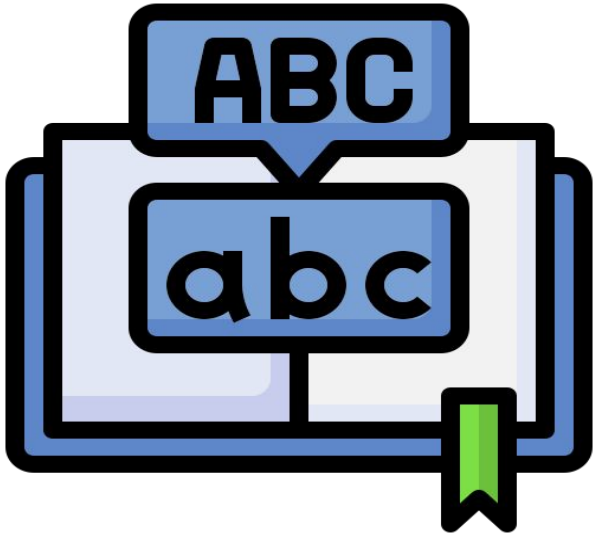
Gramática

```
# Mi primer código en Python 3

mensaje = "¡Hola, mundo!"

print(mensaje)
```

```
¡Hola, mundo!
```



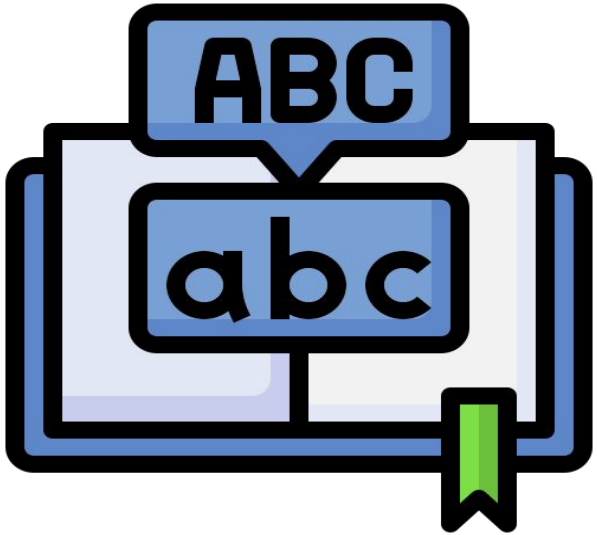
Gramática

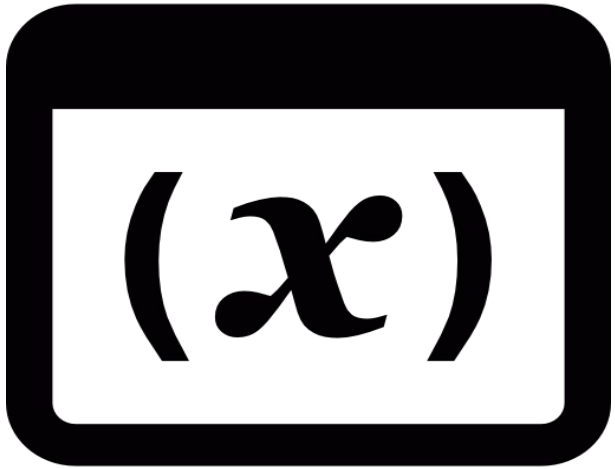
Tipos de errores

- Sintaxis
- Lógicos
- Semánticos

La depuración es el proceso de encontrar la causa del error en tu código.

1. Leer
2. Ejecutar
3. Reflexionar
4. Retroceder



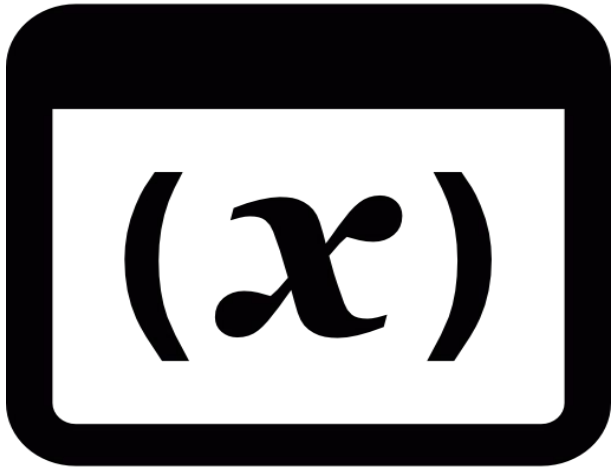


Variables, declaraciones y expresiones

Un **valor** es una de las cosas básicas con las que trabaja un programa, como una letra o un número.

Tipos de datos

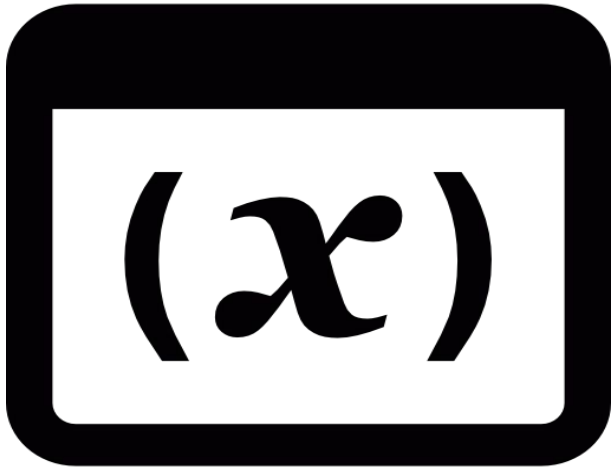
- Cadena
- Entero
- Flotante



Variables, declaraciones y expresiones

Una **variable** es un nombre que se refiere a un valor.

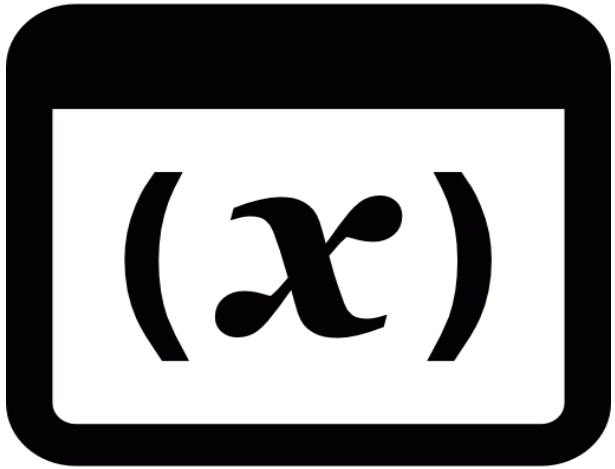
```
mensaje = "¡Hola, mundo!"
x = 1
pi = 3.1416
```



Variables, declaraciones y expresiones

Python utiliza palabras reservadas para reconocer la estructura del programa.

and	continue	finally	is	raise
def	for	lambda	return	assert
del	from	None	True	async
elif	global	nonlocal	try	await
else	if	not	while	break
as	except	import	or	with
in	class	False	pass	yield



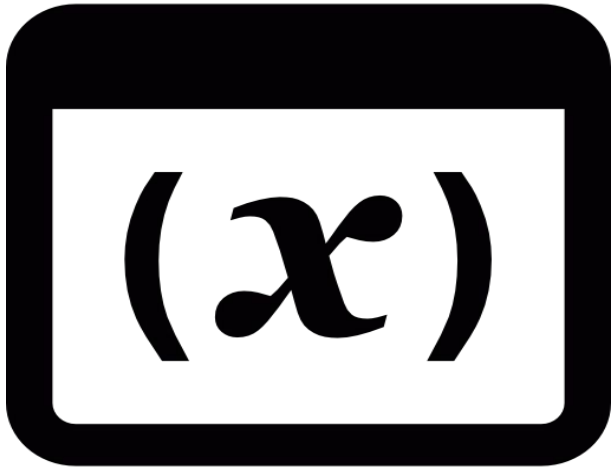
Variables, declaraciones y expresiones

Una **declaración** es una unidad de código que el intérprete de Python puede ejecutar.

```
print(1)
x = 2
print(x)
```

```
1
2
```

Un **script** es una secuencia de declaraciones.

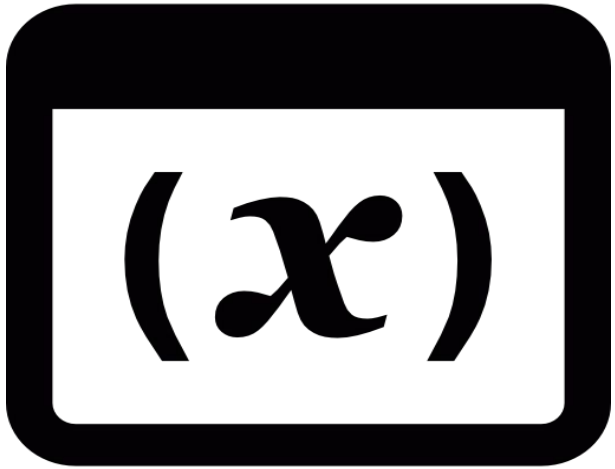


Variables, declaraciones y expresiones

Los **operadores** son símbolos especiales que representan operaciones:

- `a + b + c`
- `x - 4`
- `6 * y`
- `x ** 2`
- `32 / z`
- `64 // 3`

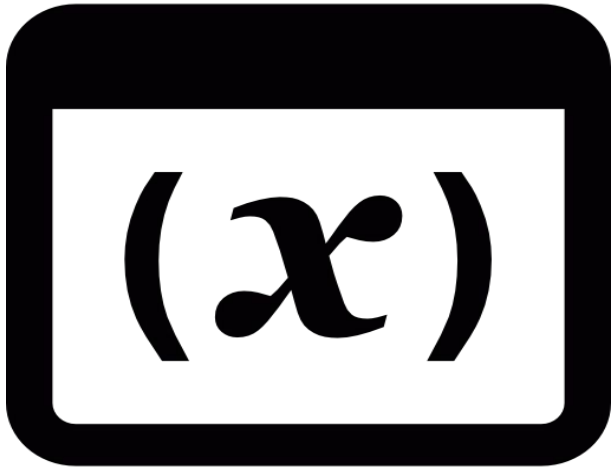
Los valores a los que se aplica el operador se llaman **operandos**.



Variables, declaraciones y expresiones

Una **expresión** es una combinación de valores, variables y operadores.

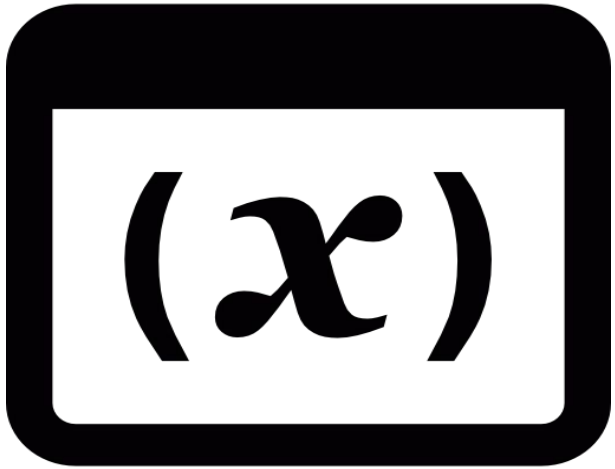
```
x = 5
x + 3
8
```



Variables, declaraciones y expresiones

Cuando aparece más de un **operador** en una **expresión**, el orden de evaluación depende de las reglas de precedencia **PEMDAS**:

1. **P**aréntesis
2. **E**xponentes
3. **M**ultiplicación y **D**ivisión
4. **A**dición y **S**ustracción

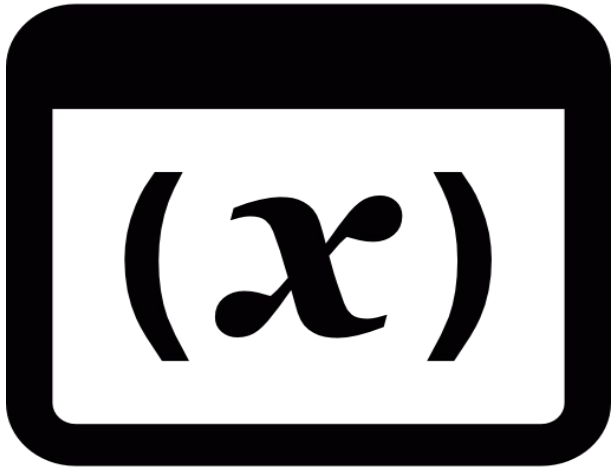


Variables, declaraciones y expresiones

El **módulo** funciona con números enteros y devuelve el valor restante de una división.

```
7 % 3
```

```
1
```

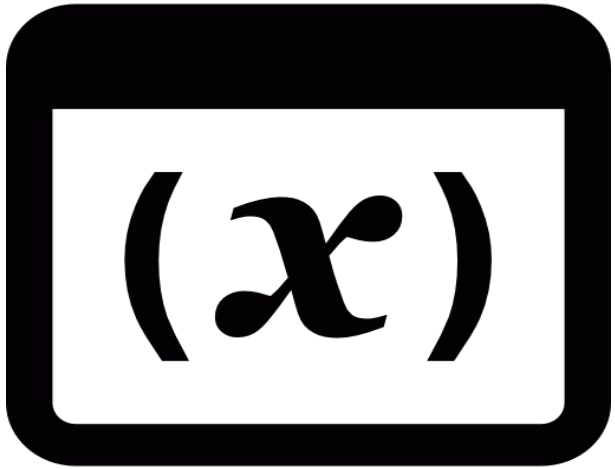


Variables, declaraciones y expresiones

Python proporciona una función incorporada llamada **input** que obtiene la entrada del teclado.

```
numero = input("Ingresa un número: ")
print(numero)
```

```
Ingresa un número: 8
8
```

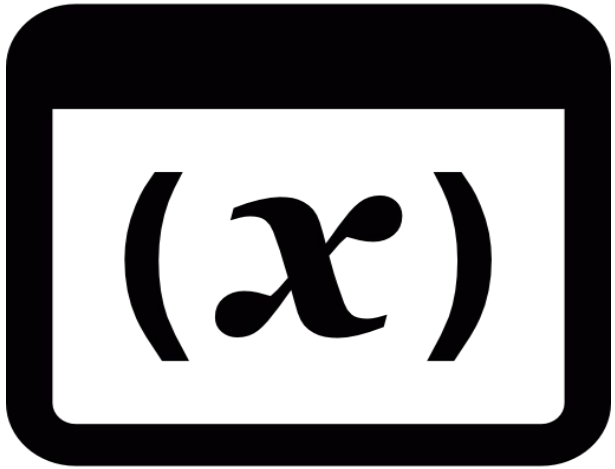


Variables, declaraciones y expresiones

Una cadena es una secuencia de caracteres.

```
fruta = 'banana
letra = fruta[1]
print(letra)
'a'
```

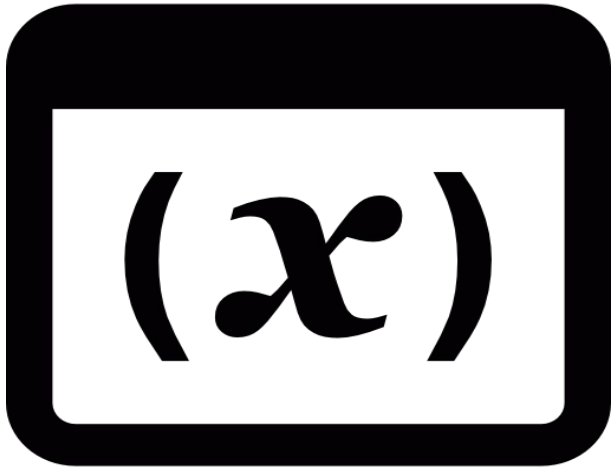
En Python, el índice es un desplazamiento desde el principio de la cadena y el desplazamiento de la primera letra es cero.



Variables, declaraciones y expresiones

Un segmento de una cadena se llama **slice.**

```
cadena = 'Python'
print(cadena[0:1])
'Py'
print(cadena[2:5])
'thon'
```



Variables, declaraciones y expresiones

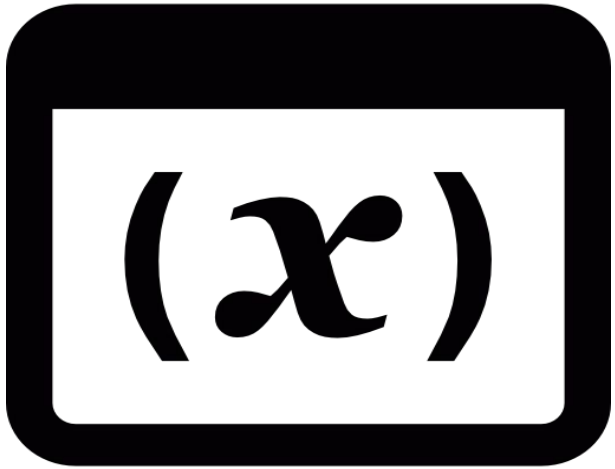
Los **métodos**, son funciones integradas en el objeto y están disponibles para cualquiera de sus instancias.

```
cadena = 'banana'

palabra = cadena.upper()

print(palabra)

'BANANA'
```



Variables, declaraciones y expresiones

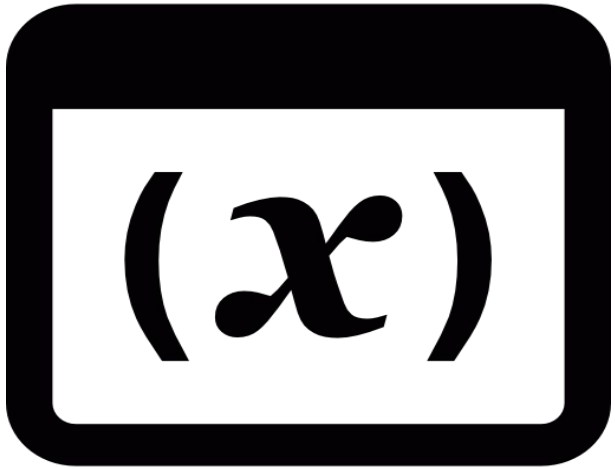
Los **métodos**, son funciones integradas en el objeto y están disponibles para cualquiera de sus instancias.

```
cadena = 'banana'

indice = cadena.find('a')

print(indice)

1
```



Variables, declaraciones y expresiones

La palabra in es un operador booleano que toma dos cadenas y devuelve True si la primera aparece como una subcadena en la segunda.

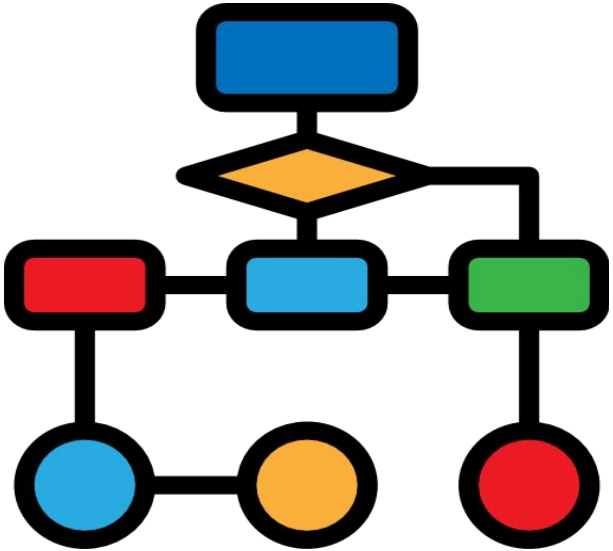
```
'a' in 'banana'
True
'seed' in 'banana'
False
```

Estructuras de control

Una expresión **booleana** es una expresión que es verdadera o falsa.

Para evaluar las expresiones utilizamos los operadores de comparación…

- !=
- >
- <
- >=
- <=



Estructuras de control

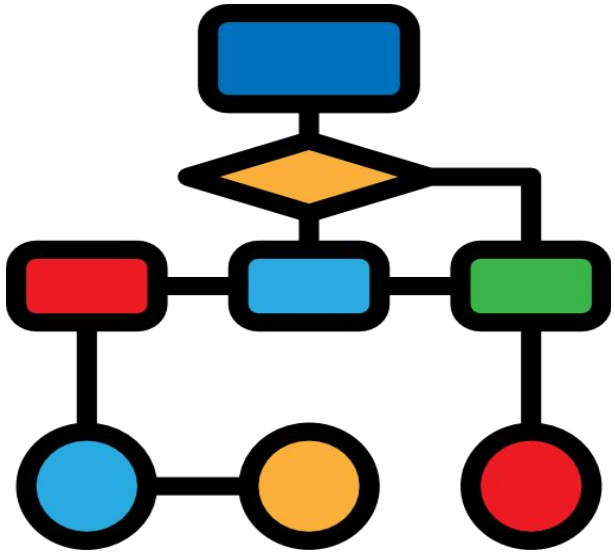
Existen tres operadores **lógicos:**

1. `and`
2. `or`
3. `not`

```
x > 0 and x < 10
```

```
x > 0 or x == 10
```

```
not(x > y)
```

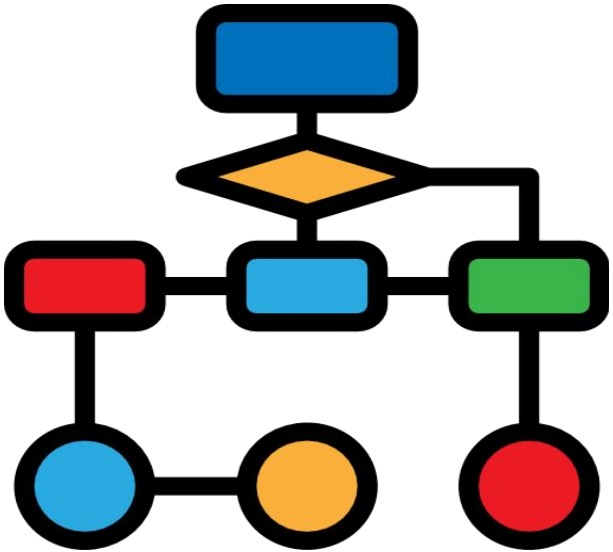


Estructuras de control

Utilizamos las declaraciones **condicionales** para verificar y cambiar el comportamiento del programa.

```
x = 8
if x > 5:
    print("Positivo")
else:
    print("Negativo")
```

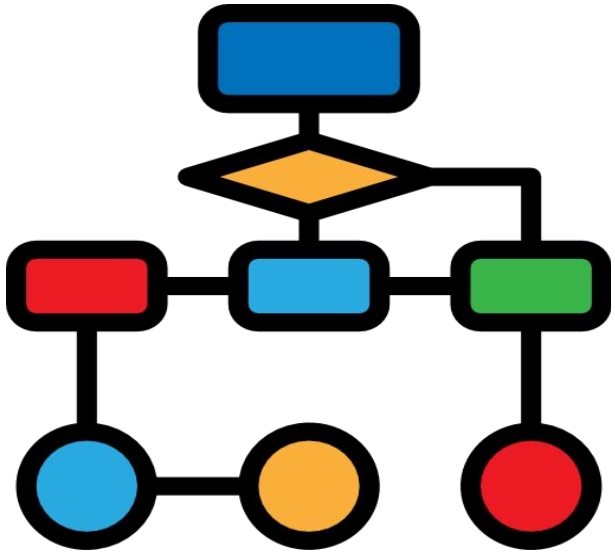
```
Positivo
```



Estructuras de control

Utilizamos las declaraciones **condicionales** para verificar y cambiar el comportamiento del programa.

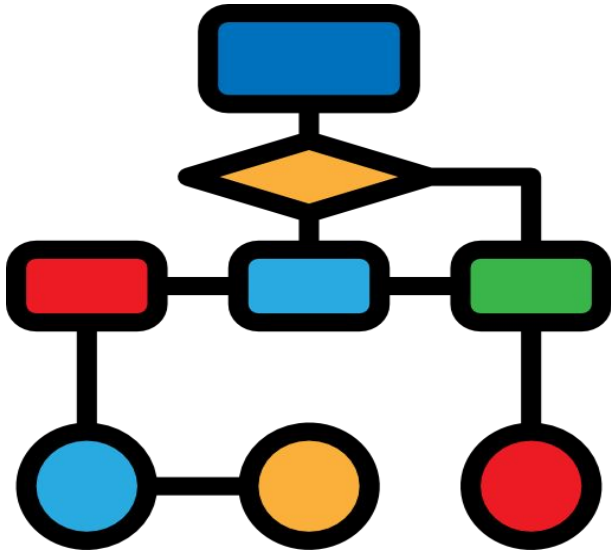
```
if x > 0:
    print("Positivo")
elif x < 0:
    print("Negativo")
else:
    print("Cero")
```



Estructuras de control

Existe una estructura de ejecución **condicional** para manejar errores esperados e inesperados llamada **try / except**.

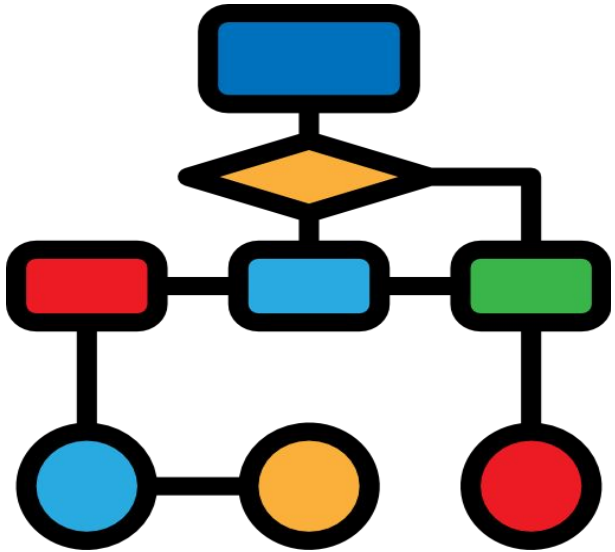
```
try:
    fahrenheit = float(inp)
    celsius = ((fahrenheit - 32.0)
    * 5.0 / 9.0)
    print(celsius)
except:
    print("Ingresa un número")
```



Estructuras de control

La instrucción **while** nos ayuda a realizar tareas repetitivas e iterar resultados.

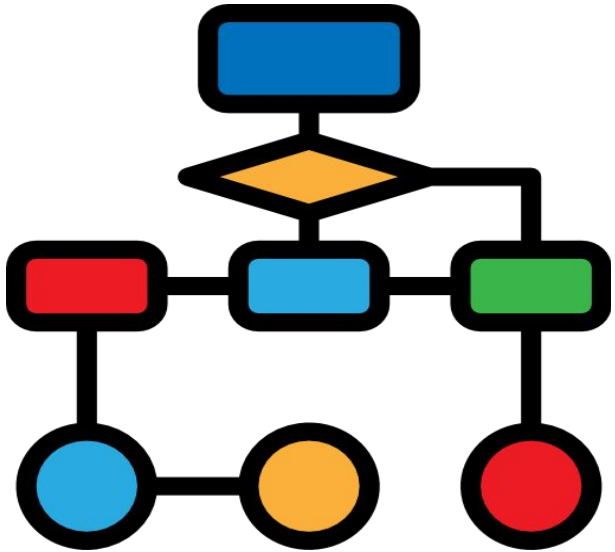
```
n = 5
while n > 0:
    print(n)
    n = n - 1
print("Boom!")
```



Estructuras de control

La instrucción **while** nos ayuda a realizar tareas repetitivas e iterar resultados.

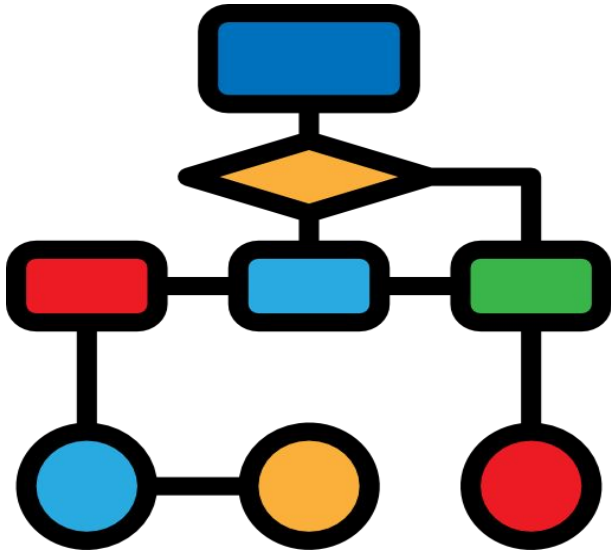
```
while True:
    dato = input('> ')
    if dato[0] == '#':
        continue
    if dato == 'bye':
        break
    print(line)
print('Adios!')
```



Estructuras de control

Cuando tenemos una lista de cosas para recorrer, podemos construir un ciclo definido usando una instrucción **for.**

```
alumnos = ['Ana', 'Luis', 'Andy']
for alumno in alumnos:
    print(f"Hola, {alumno},
    bienvenido al curso!")
print("Continuamos…")
```



Funciones

En el contexto de la programación, una función es una secuencia de instrucciones con un nombre específico que realiza una computación.

```
type(32)
<class 'int'>
```

Funciones

Pasos para crear una función

1. Definir el nombre.
2. Definir los argumentos.
3. Definir la secuencia de instrucciones que se ejecutan cuando se llama la función.

```
def imprimir_letras():
    print("But I'm in so deep")
    print("You know I'm such a
    fool for you")
```

Funciones

Pasos para crear una **función**

1. Definir el nombre.
2. Definir los argumentos.
3. Definir la secuencia de instrucciones que se ejecutan cuando se llama la función.

```
def imprimir_doble(dato):
    print(dato*2)
```

Funciones

Pasos para crear una **función**

1. Definir el nombre.
2. Definir los argumentos.
3. Definir la secuencia de instrucciones que se ejecutan cuando se llama la función.

```
def adicion(a, b):
    suma = a + b
    return suma
```

Funciones

Python proporciona una serie de **funciones integradas** que podemos usar sin necesidad de proporcionar la definición de la función.

```
max(10, 100)
100
min(10, 100)
10
len("Hola, mundo")
11
```

Funciones

También encontramos funciones que **convierten valores** de un tipo a otro.

```
str(32)
"32"
int(32.99)
32
float(32)
32.0
```

Módulos y paquetes

Un **módulo** es un archivo que contiene definiciones de funciones, clases y variables que se pueden reutilizar en otros programas.

```
import math

resultado = math.sqrt(8)
print(resultado)
2.8284
```

Manipulación de archivos

Se puede pensar en un archivo de texto como una secuencia de líneas, al igual que se puede pensar en una cadena como una secuencia de caracteres.

```
texto = open('texto.txt', 'r')
contador = 0
for linea in texto:
    contador = contador + 1
print('Total:', contador)
```

Manipulación de archivos

Podemos combinar el patrón para leer un archivo con métodos de cadena para construir mecanismos de búsqueda simples.

```
texto = open('texto.txt')
for linea in texto:
    if linea.startswith('Lorem'):
        print(linea)
```

Manipulación de archivos

Para escribir un archivo, debe abrirlo con el modo "w" como segundo parámetro.

```
cadena = "Winter is coming."
fout.write(cadena)
```

El método de escritura del archivo coloca datos en el archivo y devuelve el número de caracteres escritos.

Tuplas, listas y diccionarios

Una **lista** es una secuencia de valores de cualquier tipo.

```
[10, 20, 30, 40]
['mono', 'rana', 'gato']
['spam', 2.0 , 8, [10, 20]]
[]
```

Tuplas, listas y diccionarios

Al contrario de las cadenas, las **listas** si son **mutables**.

```
num = [17, 123]
num[1] = 5
print(num)
[17, 5]
```

Tuplas, listas y diccionarios

Python proporciona métodos que operan en listas. Por ejemplo, **append** agrega un nuevo elemento al final de una lista.

```
t = ['a', 'b', 'c']
t.append('d')
print(t)
['a', 'b', 'c', 'd']
```

Tuplas, listas y diccionarios

extend toma una lista como argumento y agrega todos los elementos.

```
t1 = ['a', 'b', 'c']
t2 = ['d', 'e']
t1.extend(t2)
print(t1)
['a', 'b', 'c', 'd', 'e']
```

Tuplas, listas y diccionarios

sort organiza los elementos de la lista de menor a mayor.

```
t = ['c', 'a', 'd', 'b']
t.sort()
print(t)
['a', 'b', 'c', 'd']
```

Tuplas, listas y diccionarios

Hay varias formas de eliminar elementos de una lista. Si conoce el índice del elemento que desea, puede usar **pop.**

```
t = ['a', 'b', 'c', 'd']
x = t.pop(3)
print(t)
['a', 'b', 'c']
print(x)
d
```

Tuplas, listas y diccionarios

Si no necesitas guardar el valor eliminado, puede usar la instrucción **del**.

```
t = ['a', 'b', 'c', 'd']
del t[3]
print(t)
['a', 'b', 'c']
```

Tuplas, listas y diccionarios

Para convertir de una cadena a una lista de caracteres, puede usar **list.**

```
cadena = 'spam'
t = list(cadena)
print(t)
['s', 'p', 'a', 'm']
```

Tuplas, listas y diccionarios

Para convertir de una cadena a una lista de caracteres, puede usar **list.**

```
cadena = 'curso de python 3'
t = cadena.split()
print(t)
['curso', 'de', 'python', '3']
```

Tuplas, listas y diccionarios

Un **diccionario** es como una lista, pero más general.

```
trad = {'one':'uno', 'two':'dos'}
print(trad)
{'one':'uno', 'two':'dos'}
```

Puede pensar en un **diccionario** como un mapeo entre un conjunto de índices (denominados claves) y un conjunto de valores.

Tuplas, listas y diccionarios

Los **diccionarios** tienen un método llamado **get** que toma una clave y un valor predeterminado.

```
palabra = 'brontosaurus'
d = dict()
for c in palabra:
     if c not in d:
          d[c] = 1
     else:
          d[c] = d[c] + 1
print(d)
```

Tuplas, listas y diccionarios

Una **tupla** es una secuencia de valores muy parecida a una lista. Sintácticamente, una **tupla** es una lista de valores separados por comas.

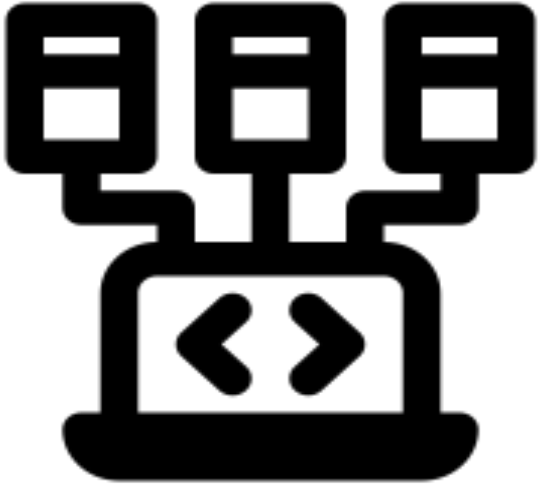
```
t = ('a', 'b', 'c', 'd', 'e')
```

```
a, b = b, a
```

Tuplas, listas y diccionarios

Los diccionarios tienen un método llamado elementos que devuelve una lista de **tuplas**, donde cada **tupla** es un par clave-valor:

```
d = {'b':1, 'a':10, 'c':22}
t = list(d.items())
print(t)
[('b', 1), ('a', 10), ('c', 22)]
```



Programación orientada a objetos

Un objeto contiene datos, métodos y funciones integradas disponibles para cualquier instancia del objeto.

```
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre

    def saludar(self):
        print(f"Hola, mi nombre
            es: {self.nombre}")
persona = Persona("Lupe")
persona.saludar()
Hola, mi nombre es Lupe
```

Herramientas esenciales

Librerías y paqueterías

Web Scraping

Análisis de datos

APIs

Django