Visión Computacional

Tutorial 1: Introducción a Python

Jose Laruta

Diplomado en Sistemas Robóticos avanzados - Unifranz - Octubre 2021

Agenda

1. Introducción Python
2. Entorno de desarrollo
3. El intérprete de Python
4. Variables
5. Funciones
6. Sentencia condicional `if`
7. Bucles
8. Colecciones
9. OOP

Python

Lenguaje de programación interpretado, tiene como filosofía la **simplicidad y legibilidad**. Actualmente es uno de los lenguajes más populares y usados en la industria.

Python

- Sintaxis simple y limpia.
- Programación multiparadigma.
- Tipado dinámico.
- Lenguaje interpretado.
- Open source.

¿Dónde se usa Python?

- Desarrollo Web.
- Ciencia de Datos.
- Visión Artificial.
- IA, ML, DL.
- Investigación.
- Automatización.

Lenguajes Compilados

Se tiene un archivo de **código fuente** que es procesado por un programa llamado el **compilador** que transforma el código en un archivo **ejecutable**.

- C/C++
- Java
- Rust

El compilador puede analizar y optimizar el código fuente.

Lenguajes Interpretados

No existe un compilador. Se tiene un programa **intérprete** que ejecuta los comandos de forma **inmediata***.

- Python
- Lua
- Php

Se puede iterar más rápido sin esperar el proceso de compilación.

Entorno de desarrollo

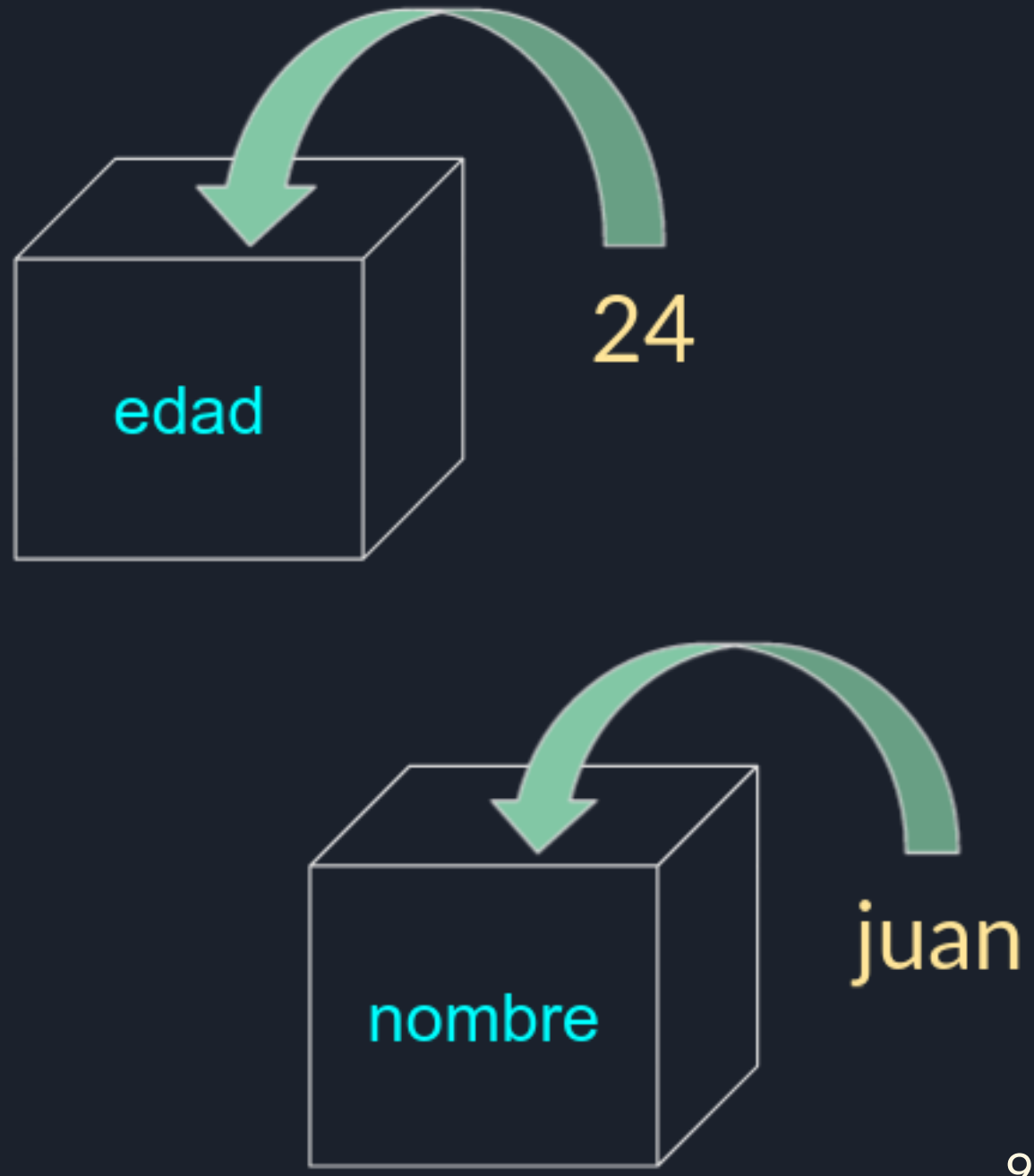
Instrucciones en: https://github.com/tabris2015/computer-vision-unifranz2021

Variables

Una variable almacena información de un tipo definido.

Existe en la memoria RAM y se puede modificar una vez creada.

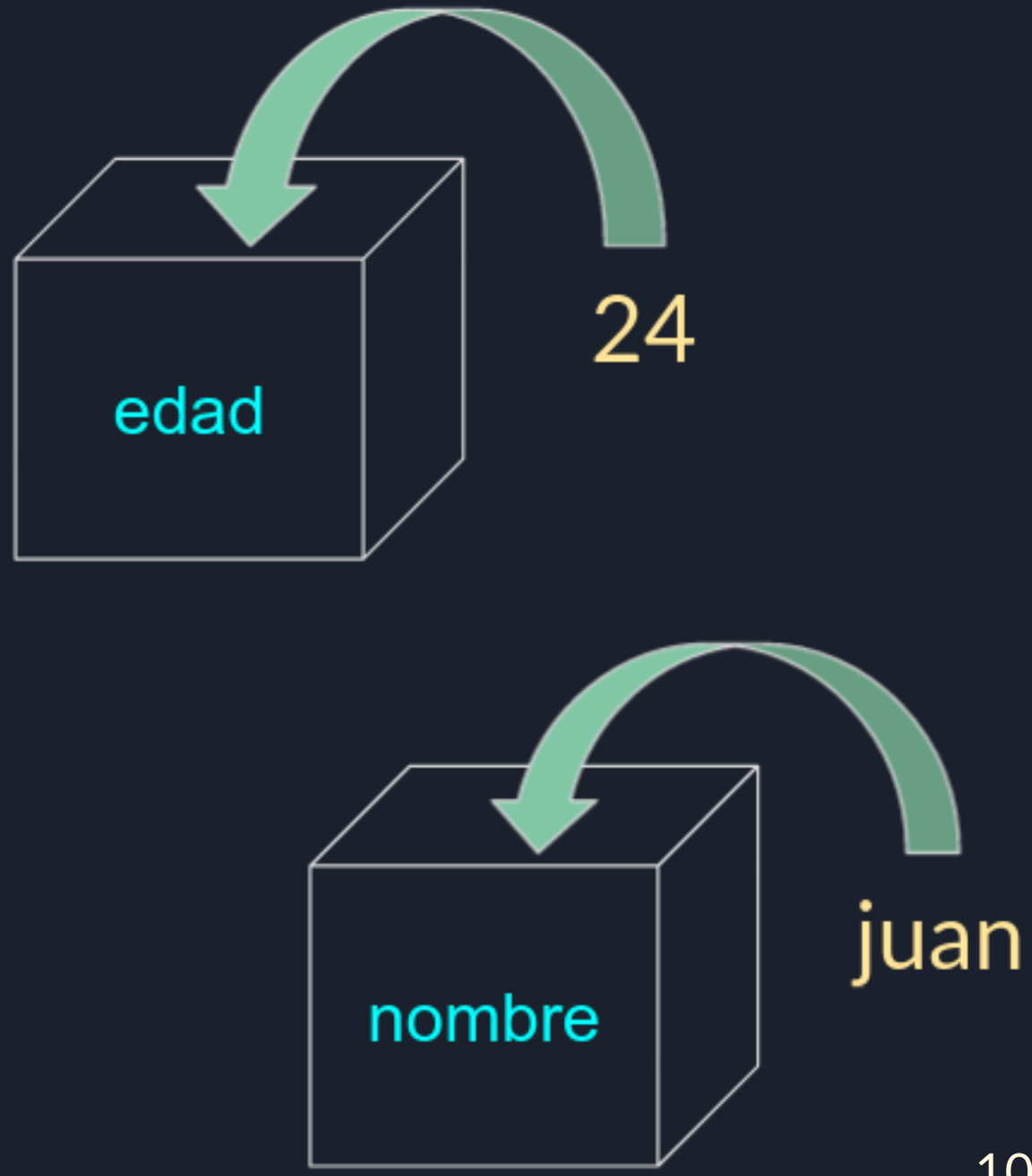
Es un *contenedor* con una etiqueta.



Variables

En Python las variables se declaran con el operador de asignación `=`.

```
edad = 24
nombre = "juan"
pi = 3.1415962
```



Tipos primitivos

- Enteros `int`.
- Flotantes `float`.
- Cadenas o strings `str`.
- Booleanos `bool`.

Operadores aritméticos

- Adición `+`
- Substracción `-`
- Multiplicación `*`
- División `/`
- División entera `//`
- Módulo o resto `%`
- Exponenciación `**`

El intérprete de Python

Ingresar al intérprete usando el comando

```
$ python
```

Cadenas de Caracteres

Los Strings representan texto.

En Python son objetos con muchas funcionalidades.

Se puede declarar una cadena literal usando comillas dobles o simples:

```
nombre = 'juan'
apellido = "perez"
```

Operaciones con cadenas

Se puede concatenar cadenas usando el operador `+`:

```
prefijo = "hiper"
nombre = "maxi"
prefijo + nombre
# Salida: 'hipermaxi'
```

Se puede repetir una cadena usando el operador `*`:

```
"sopa" * 4
# Salida: 'sopasopasopasopa'
```

Métodos del objeto String

```
cadena = " hola amigos   "
```

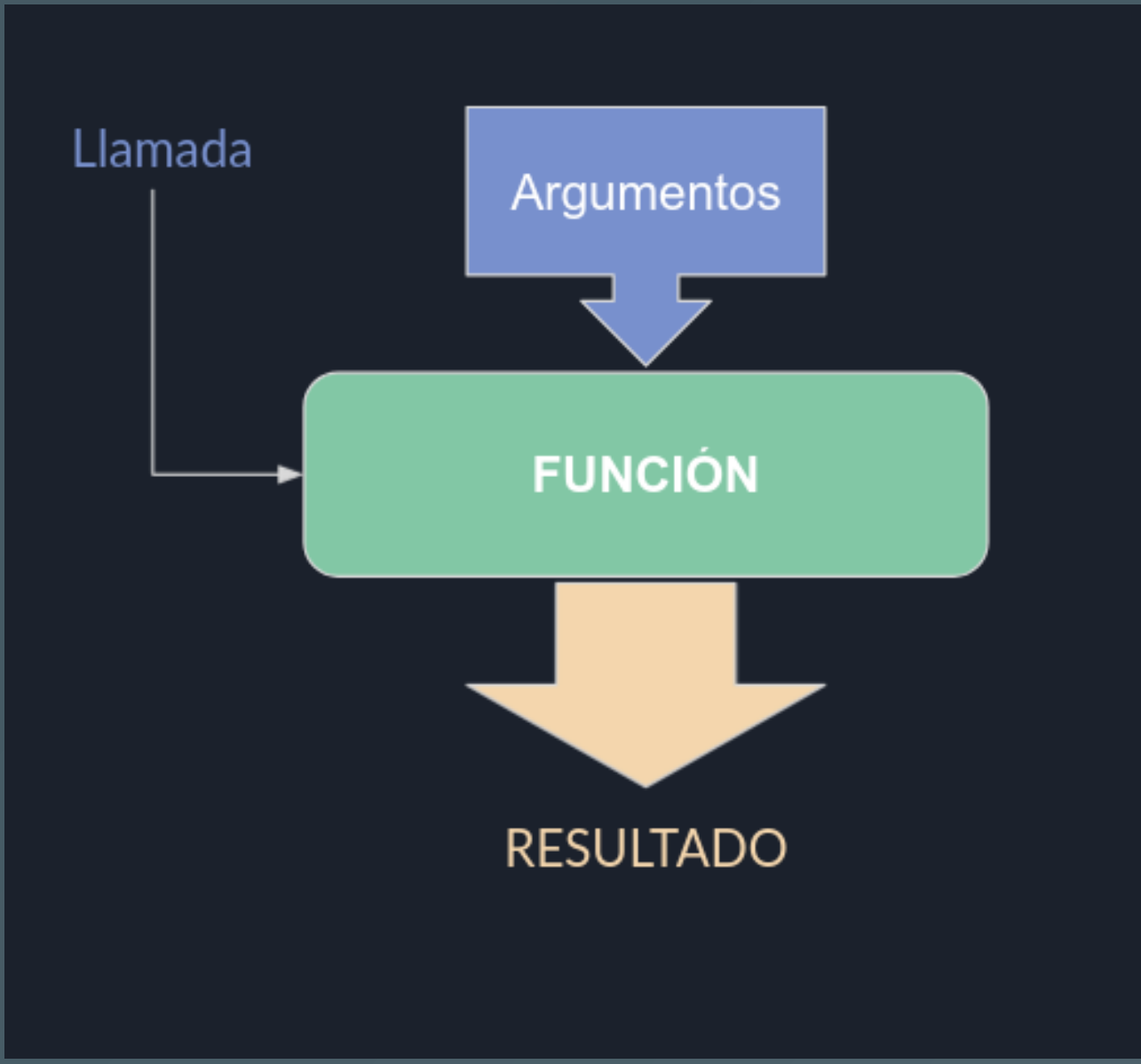
- `cadena.strip()`
- `cadena.lower()`
- `cadena.upper()`
- `cadena.replace("hola", "adios")`
- `cadena.split()`
- y muchos más...

Funciones

Una función es una porción de código que se ejecuta solamente cuando se realiza una *llamada*.

Puede recibir datos como parámetros.

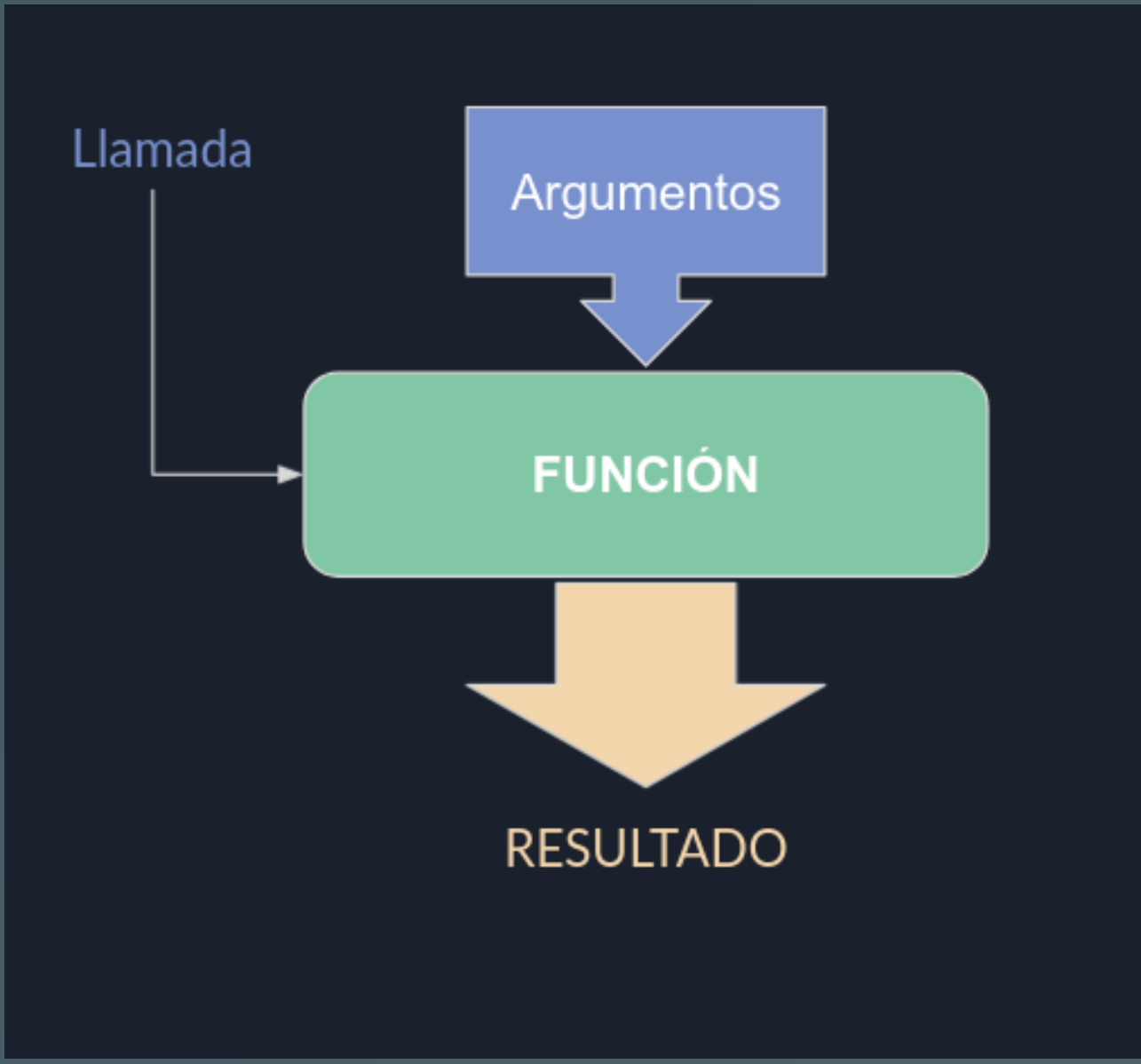
Puede retornar valores.



Funciones

Generalmente se tienen dos etapas:

- **Declaración**: Se define la interfaz y el comportamiento de la función.
- **Llamada**: Se realiza la ejecución del código fuente de la función.



Funciones

La declaración de funciones se realiza usando el keyword `def`.

```
# declaracion de una funcion
def mi_funcion(arg1, arg2):
    resultado = arg1 + arg2
    return resultado

# llamada de la funcion
mi_funcion(70, 9)
# salida: 79
```

Funciones

La declaración de funciones se realiza usando el keyword `def`.

```
# declaracion de una funcion con type hints
def mi_funcion(arg1: int, arg2: int) -> int:
    resultado = arg1 + arg2
    return resultado

# llamada de la funcion
mi_funcion(70, 9)
# salida: 79
```

Ejecución condicional

En python se puede definir una ejecución condicional usando la sentencia `if`.

```
if condicion:
    # cuando la condicion se cumple
else:
    # cuando la condicion NO se cumple
```

Operadores de comparación

Los operadores de comparación comparan dos valores y se evalúan a un valor de tipo `bool`.

- Igualdad `==`
- Desigualdad `!=`
- Mayor que `>`
- Menor que `<`
- Mayor o igual `>=`
- Menor o igual `<=`
- Identidad `is`

Operadores lógicos

Los operadores lógicos se usan para combinar sentencias condicionales.

- `and`
- `or`
- `not`

Ejecución condicional

Se pueden agregar condiciones adicionales con la sentencia `elif`

```
edad = 18

if edad < 15:
    print("muy joven")
elif edad > 70:
    print("muy viejo")
else:
    print("OK")

# Salida: 'OK'
```

Bucles

En Python se tienen dos sentencias para ejecutar bucles.

- `while` ejecuta un conjunto de comandos de forma repetida *mientras* se cumpla una condición.
- `for` ejecuta un conjunto de comandos de forma repetida para una secuencia definida.

while

```
contador = 4
while contador > 0:
    print(contador)
    contador = contador - 1

#Salida:
# 4
# 3
# 2
# 1
```

while

```
intentos = 4
while intentos > 0:
    print(intentos)
    if not conexion_db():
        # Terminacion del bucle con break
        print("base de datos desconectada")
        break
    intentos -= 1
```

for

En python, el bucle `for` se usa para iterar sobre una **secuencia**.

```
# range(5) genera la secuencia 0 1 2 3 4
for n in range(5):
    print(n)
#Salida:
# 0
# 1
# 2
# 3
# 4
```

for para strings

Un string es una secuencia ordenada de caracteres.

```
saludo = "hola"
# se itera para cada caracter de la cadena
for c in saludo:
    print(c)
#Salida:
# h
# o
# l
# a
```

Colecciones

Son objetos especiales que sirven para agrupar varios elementos en un solo objeto o *contenedor*.

- Listas
- Tuplas
- Diccionarios
- Conjuntos

Listas

Colección ordenada de elementos del mismo o distinto tipo.

```
lista_compras = ["papa", "zanahoria", "carne"]
```

Listas

Se puede acceder a sus elementos en base al índice:

```
lista_compras = ["papa", "zanahoria", "carne"]
print(lista_compras[0])
# Salida: 'papa'
```

Listas

Se puede modificar los elementos:

```
lista_compras = ["papa", "zanahoria", "carne"]
print(lista_compras[0])
# Salida: 'papa'
lista_compras[2] = "pescado"
print(lista_compras)
# Salida: ['papa', 'zanahoria', 'pescado']
```

Tuplas

Similar a una lista, sin embargo no se puede modificar una vez definida

```
tupla_compras = ("papa", "zanahoria", "carne")
print(tupla_compras[0])
# Salida: 'papa'
tupla_compras[2] = "pescado"
# Error!
```

Diccionarios

Colección de elementos *clave-valor* o *key-value*.
El *key* es un objeto inmutable y el *value* puede ser cualquier objeto de python.

```
estudiantes = {"sosa": 67, "choque": 90}
```

Diccionarios

Se puede acceder a través del *key*

```
estudiantes = {"sosa": 67, "choque": 90}
print(estudiantes["choque"])
# Salida: 90
```

Diccionarios

Se puede modificar un elemento a través del *key*

```
estudiantes = {"sosa": 67, "choque": 90}
print(estudiantes["choque"])
# Salida: 90
estudiantes["sosa"] = 78
print(estudiantes)
# Salida: {'sosa': 78, 'choque':90}
```

Diccionarios

Se puede agregar un nuevo elemento:

```
estudiantes = {"sosa": 67, "choque": 90}
print(estudiantes["choque"])
# Salida: 90
estudiantes["sosa"] = 78
print(estudiantes)
# Salida: {'sosa': 78, 'choque':90}
estudiantes["perez"] = 100
print(estudiantes)
# Salida: {'sosa': 78, 'choque': 90, 'perez': 100}
```

Diccionarios

Un valor puede ser otro diccionario

```
estudiante = {"nombre": "juan", "notas": {"mat": 50, "fis": 70}}
print(estudiante["notas"]["mat"])
# Salida: 50
```

Programación orientada a objetos

Los objetos contienen datos y comportamiento aislados de otros objetos.

Ayuda en la organización, legibilidad, reusabilidad y mantenibilidad del código.

Programación orientada a objetos

- Encapsulamiento
- Abstracción
- Herencia
- Polimorfismo

Elementos de POO

- Clases
- Objetos
- Métodos
- Atributos

Clases en Python

En python podemos implementar la orientación a objetos a través de las clases:

```
class Carro():
    marca = "subaru"
    def encender(self):
        # codigo para encender el carro
    def avanzar(self):
        # codigo para arrancar
```

Clases en Python

Se cuenta con el método `__init__` que se ejecuta al instanciar un objeto:

```
# Declaracion de la clase
class Estudiante:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
    def saludar(self):
        print(f"hola, soy {self.nombre} y tengo {self.edad} años")

est = Estudiante("juan", 40)
est.saludar()
# Salida: 'hola, soy juan y tengo 40 años'
```