# Tipos de datos en Python

# Contenidos:

- Tipos de Datos Estándar o Nativos
- Identificadores: Variables / Constantes
- Operadores: Asignación, Aritmeticos
- Tipo de una expresión y Coerción
- Estructura General de un programa Python
- Datos estructurados: Listas, Tupla, Diccionario y Conjuntos(Set)

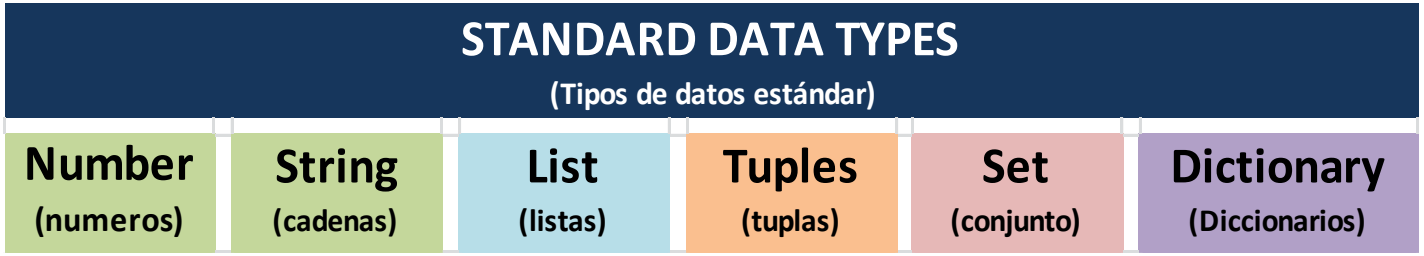
# Tipos de datos standard o nativos

# Tipos de datos estándar/nativos en Python: Analogía

- Los **tipos de datos** sirven para identificar que tipo de valor se está utilizando.
- Por ejemplo, si tenemos que hablar acerca de estudiante, debemos saber su nombres y apellidos, DNI, edad y estatura .
  - El nombre y apellido son **palabras**.
  - El DNI y la edad es un **número entero**.
  - La estatura un **número decimal**.
- En este ejemplo tenemos tres tipos de datos, palabras, números enteros y números decimales.

# Tipos de datos estándar/nativos en Python

- Los datos que procesa la computadora, son básicamente de tres tipos: enteros, decimales y letras.
- La computadora siempre procesa datos y los transforma en información.
- En Python existen 6 tipos de datos estándar:
  - Tipos de datos simples como los enteros y decimales
  - Tipos de datos complejos como las listas, tuplas, conjuntos y diccionarios.



# Tipos de datos estándar/nativos en Python: Ejemplos

| TYPE OF DATA | PYTHON TYPE NAME | EXAMPLE LITERALS |
|---|---|---|
| Integers | `int` | `-1, 0, 1, 2` |
| Real numbers | `float` | `-0.55, .3333, 3.14, 6.0` |
| Character strings | `str` | `"Hi", "", 'A', '66'` |

{'key': 'value'}

**STANDARD DATA TYPES**
(Tipos de datos estándar)

| Number (numeros) | String (cadenas) | List (listas) | Tuples (tuplas) | Set (conjunto) | Dictionary (Diccionarios) |
|---|---|---|---|---|---|

```
# empty list
my_list = []

# list of integers
my_list = [1, 2, 3]

# list with mixed datatypes
my_list = [1, "Hello", 3.4]
```

( [1, 2], (3, 4), "Tuple" )

```
# set of integers
my_set = {1, 2, 3}
print(my_set)

# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

# Identificadores

# Identificadores

- Usados para dar nombre a los diferentes objetos que componen un programa:
  - variables
  - funciones
  - clases
  - módulos
  - paquetes
- Por ejemplo: altura, base, input

# Identificadores: Sintaxis

- **Definición**: Un *identificador* es una secuencia de caracteres *alfanuméricos*, el primero de los cuáles debe ser *alfabético*.
- **Gramática BNF** (Backus/Naur Form):

```
identifier ::=  (letter|"_") (letter | digit | "_")*
letter     ::=  lowercase | uppercase
lowercase  ::=  "a"..."z"
uppercase  ::=  "A"..."Z"
digit      ::=  "0"..."9"
```

# Identificadores: El uso de mayúsculas y minúsculas

- Python es un lenguaje ***case sensitivo***, es decir hace distinción entre las letras mayúsculas de minúsculas.
- Por ejemplo un programa utiliza los identificadores siguientes:

  - **casa**
  - **CASA**
  - **Casa**
  - **cAsA**

**Estos identificadores son distintos. Se diferencia por el uso de las letras mayúsculas y minúsculas**

# Identificadores: Palabras reservadas

- También llamadas ***keywords***
- Son ciertos **identificadores cuyo uso se prohíbe**
- Los **utiliza el lenguaje con propósitos particulares** como parte de las instrucciones.
- En el ejemplo mostrado se utilizan varias palabras reservadas

```
6  def main():
7      x = int(10)
8      print ("Hola Mundo Cruel")
9      print (str(10*10))
10
11 if __name__ == '__main__':
12     main()
13
```

**PALABRAS RESERVADAS**

1. def
2. int
3. print
4. str
5. if
6. __name__

# Identificadores: Palabras reservadas

```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

# Variables y constantes

# Variables y constantes en Python

- **LAS VARIABLES**

  Son contenedores (espacios de memoria) que permiten almacena un valor con la posibilidad de modificarlos.

- **LAS CONSTANTES**

  Son contenedores (espacios de memoria) permiten almacenar un valor pero no tiene posibilidad de modificarlo.

- **VARIABLES Y CONSTANTES: Recomendaciones**

  Las variables y constantes deben poseer un nombre único dentro del programa.

  **Declarar** (**en Python no es obligatorio**) las variables antes de poder utilizarlas por lo que tienen que especificar a que tipo de dato al cual pertenecen.

# Nombres de variables y constantes

- Recomendaciones para asignar el nombre:
  - El primer carácter debe ser una letra o el signo de subrayado "_", los demás pueden ser letras, el signo del subrayado o dígitos.
  - **NO** se pueden utilizar caracteres especiales en el nombre como por ejemplo (ñ, ?, , , %, etc.)
    - **Año = 1997 # Error en esta instrucción**
    - **% = 0.85 # Error en esta instrucción**
  - **NO** puede coincidir con las palabras reservadas del lenguaje, por ejemplo int, if, elif, import, from etc.
    - **import** = "Usa" **# Error en esta instrucción**
    - **from** = "origen" **# Error en esta instrucción**

# Nombres de variables y constantes: Ejemplos

<Nombre variable> = <valor>

Por ejemplo:

variableEntera = 10

Estatura = 1.72

mi_primera_letra= ‘a’

estado_civil=

_i18n_ = “US”

<Nombre variable> = tipo(<valor>)

Por ejemplo:

variableEntera = int(10)

Estatura = float(1.72)

_estaVivo_=str(“SI”)

estadoCivil= ‘a’

i_18n_=

# Operadores

# Operadores: Tipos

- **Asignación**
- **Aritméticos**
- **Relación (Se revisara mas adelante)**
- **Lógicos (Se revisara mas adelante)**

# Operadores: Operador de Asignación

- **Permite** dar un valor a una variable o constante.
- **Utiliza** el símbolo igual (=) como operador de asignación.
- ***La asignación siempre se realiza de derecha a izquierda***.

```
x = 58;
x = 198.54;

valor = 98.55684;
valor = valor * 0.0;

letra = ‘A’;
letra = letra + 2;
letra = ‘X’;
letra = 65;
```

```
x = int(58)
x = float(198.54)

valor = float(98.55684)
valor = valor * 0.0;

char letra = char(‘A’)
letra = letra + 2;
letra = ‘X’;
letra = 65;
```

# Operadores: Operadores Aritmeticos

| Símbolo | Significado | Ejemplo | Resultado |
|---|---|---|---|
| + | Suma | a = 10 + 5 | a es 15 |
| - | Resta | a = 12 - 7 | a es 5 |
| - | Negación | a = -5 | a es -5 |
| * | Multiplicación | a = 7 * 5 | a es 35 |
| ** | Exponente | a = 2 ** 3 | a es 8 |
| / | División | a = 12.5 / 2 | a es 6.25 |
| // | División entera | a = 12.5 / 2 | a es 6.0 |
| % | Módulo | a = 27 % 4 | a es 3 |

A excepción de la operación de modulo %, que se aplica a datos enteros, todas las operaciones dan resultados:

- Del mismo tipo que los operandos si ambos son de mismo tipo ó
- Del tipo de mayor rango si los operandos son de tipos distintos. (1).

# Operadores: Operadores aritméticos (ejemplos de uso)

| Operación | Igual a | Resultado |
|---|---|---|
| 15 / 2 + 3.0 * 2 | 7 + 6.0 | 13.0 |
| 15 / 2.0 + 3.0 * 2 | 7.5 + 6.0 | 13.5 |
| (3 – 4.7) * 5 | -1.7 * 5 | -8.5 |
| 1 + 7 % 3 | 1 + 1 | 2 |

- Se puede emplear paréntesis para agrupar datos y especificar un cálculo.
- El orden en que se evalúa la expresión se puede especificar utilizando paréntesis, o se asume el orden de precedencia que se indica a continuación:
  - Paréntesis
  - *, /, %
  - +, -

# Operados: Operadores Aritmeticos (Expresiones)

- Se construyen usando los operadores + - * // / % **:
- Ejemplos de expresiones
  - 4 + 2
  - (a + 3) / x
  - (a + 8) / (b + 2.0)

```
def operaciones():
    print("Operaciones Aritmeticas Basicas")


    print("Primer Numero? ")
    x = float(input())

    print("Segundo Numero(debe ser diferente de cero)? ")
    y = float(input())

    print ("suma=> " + str(x+y))
    print ("resta=> " + str(x-y))
    print ("multiplicacion=> " + str(x*y))
    print ("division=> " + str(x/y))
    print ("Modulo=> " + str(x%y))

#
operaciones()
```

# Coerción y tipado

# Coerción: Conversion implícita de numeros

- Si en una expresión aparece al menos un operando real, todo los otros operandos se transforman a real.
- Si se quiere evaluar **4 + 5.3** se hace **4.0 + 5.3**
- Esta transformación se llama *coerción*: un valor es forzado a cambiar de tipo automáticamente.
- Python provee operadores(funciones) de conversión explícitos:
  - int()
  - float()
  - long()
  - bool()
  - complex()

# Tipo de una expresión

- El tipo de una expresión puede ser consultado con el operador **type()**

```
>>> type(4+3)
<type 'int'>
>>> type (4.0 - 8 * 3)
<type 'float'>
>>> type(99999999999999999999999999)
<type 'long'>
```

# Lectura y Escritura de datos

# Lectura/Escritura de datos

- Cuando las variables necesitan de datos que el usuario debe ingresar desde el teclado, estamos frente a una operación de lectura de datos

- Cuando el usuario necesita mirar o presentar los resultados o valores de las variable estamos frente a un mecanismo de Lectura de datos.

# Lectura/Escritura: ejemplos

```python
def evaluacionNumeros():
    print("Cual es tu Edad? ")
    edad = int(input()) #leer un numero entero

    print("Cual es tu talla?: ")
    talla = float(input()) #lee un flotante

    print("Cual es tu Nombre? ")
    nombre = raw_input() #leer una cadena

    print("Tipo de las variables");
    print("edad es")
    print(type(edad))

    print("talla es")
    print(type(talla))

    print("nombre")
    print(type(nombre))

#
evaluacionNumeros()
```

```python
def convertidorNumeros():
    print("Ingresa un numero entero")
    numEntero = int(input())

    print("Ingresa un numero flotante(decimal)")
    numFloat = int(input())

    print("numero entero ingresado" + str(numEntero))
    print("numero flotante ingresado" + str(numFloat))
    print("nuevos numeros");

    nuevoEntero = int(numFloat)
    nuevoFloat = float(numEntero)
    print("Flotante convertido a entero=> " +str(nuevoEntero))
    print("Entero convertido a float=> " +str(nuevoFloat))


#
convertidorNumeros()
```

# Estructura de un Script Python

# Estructura General de un Programa

- Lineas de Inicializacion
- Documentacion del Script
- Importacion de Modulos
- Declaracion de variables Globales
- Declaracion de Clases
- Declaracion de Funciones
- Cuerpo Principal

```
1  #/usr/bin/env  python      #Linea de inicialización
2  "Modulo de ejemplo"        #Documentacion del script
3
4  import sys
5  import os                  #Modulos que se importan
6
7  debug= True                #Declaracion de variables Globales
8
9  class Ejemplo(Object):
10     "clase Ejemplo"        #Declaracion de Clase
11
12 def main():
13     "Main"                 #Declaracion de Funciones
14     obj= Ejemplo()
15
16 if __name__=='__main__':   #Cuerpo Principal
17     main()
```

# Estructura Simplificada de un Programa (Sugerida)

| |
|---|
| Documentacion del Script |
| Declaracion de Funciones |
| Cuerpo Principal |

```
#-------------------------------------------------------------------------
# Name:        Calcular Area de un Triangulo
# Purpose:     Mostrar el uso
#
# Author:
#
# Created:     08/04
# Copyright:   (c)
# Licence:     free
#-------------------------------------------------------------------------


def main():
    PI = 3.14159
    PI = 2*3.14159
    print("PI=> "+str(PI))

    a = int(10)
    b = float(3.14159)
    c = str('T')
    print("X=> " + str(a))
    print("y=> " + str(b))
    print("y=> " + str(c))


if __name__ == '__main__':
    main()
```

- Existen varia formas para elaborar scripts(programas en Python)
- Nuestro curso utilizaremos ambas estructuras, pero iniciaremos con la estructura simplifica (sugerida) mostrada.

# Ejercicios

# Ejemplo 1

**Escriba un programa en Python que permita calcular el área de un circulo.**

**Ejemplo:**

**Ingrese el valor del radio: 10**

**Area  Circulo => 314.159**

```
def calcularAreaCirculo():
    radio = float(0.0)  #declara la variable radio de tipo float con valor 0
    PI = float(3.14159) #declara la variable PI con valor3.14159

    print("Radio Circulo? ") #imprime la cadena "Radio Circulo" en la pantalla
    radio = input() #lee el valor del radio desde el teclado y lo almacena en radio

    area = PI * radio * radio #calcula el valor del area y lo almacena en una variable llamada area

    #str(area) convierte a cadena el valor de area

    print("Area Circulo=> " +str(area)) #imprime la cadena "Area Circulo" concatenada con el
                                        #valor del area convertido a cadena por str(area)
#
calcularAreaCirculo() #invoca a la funcion calcularAreaCirculo()
```

# Ejemplo 2

**Escriba un programa en Python que permita calcular el volumen de un cubo.**

**Ejemplo:**

**Longitud del lado del cubo?: 10**

**Volumen: 1000**

```python
def calcularVolumenCubo():
    #declara variables
    lado = float(0)
    volumen = float(0)

    #entrada de datos
    lado = input("Longitud del lado del cubo? ")

    #proceso
    volumen = lado * lado * lado

    #salida
    print("Volumen: " + str(volumen))


calcularVolumenCubo()
```

# Datos estructurados en Python

# List (Lista)

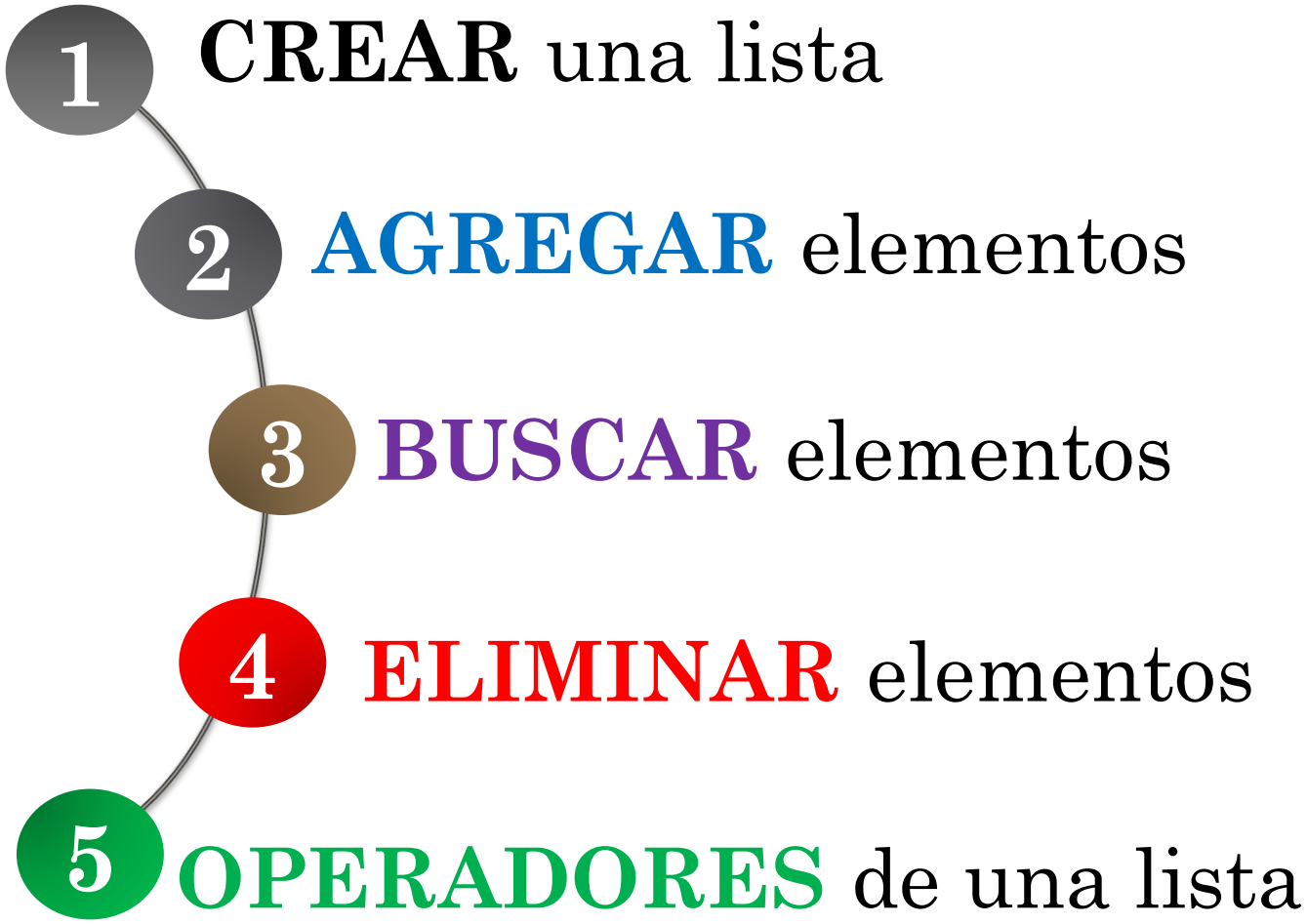
```
["Python", 11, "C", "R"]
```

# List (Lista) : Conceptos

- En Python una **LISTA** es:
  - Un **tipo de dato compuesto** puede contener cualquier numero de ítems del mismo o diferente tipo.
  - Una **secuencia de valores separados por coma**.
  - **Cada uno de ellos está numerado**, a **partir de cero**:
    - El **primero** se numera **0**
    - El **segundo** se numera con **1**
    - El **tercero** se numera con **2**, etc.

```
[ "Pyhton" , 11 , "C" , "R" ]
    0        1     2     3
```

# List (Lista): Operaciones sobre una Lista



# List (Lista): CREACION de una lista

- Una lista se crea:
  - Colocando todos los items (elementos) dentro de **CORCHETES [ ]**, separados por **coma**.
  - Invocando a la función **list().**

```
[ ]
[ "Pyhton" , 11 ,"C" ,"R" ]
list()
```

# List (Lista): CREACIÓN de una lista - ejemplo

```
1
2  def crearListas():
3      # crea una LISTA VACIA
4      lista1 = []
5
6      # crea una LISTA DE ENTEROS
7      lista2 = [1, 2, 3]
8
9      # crea una lista con TIPO DE DATOS MIXTOS
10     lista3 = [1, "Hello", 3.4]
11
12     #crea una lista ANIDADA
13     lista4 = ["mouse", [8, 4, 6], ['a']]
14
15     #crea una lista con la funcion list()
16     #la lista5 no tiene elementos
17     lista5 = list()
18
19     #imprimir
20     print(lista1)
21     print(lista2)
22     print(lista3)
23     print(lista4)
24     print(lista5)
25 #
26 crearListas()
```

# List (Lista): AGREGAR a una lista

- Formas :
  - **APPEND**: Agregar elementos al final de la lista.
  - **INSERT**: Agrega un elemento simple en la lista, en la posición especificada por el indice.
  - **EXTENDS**: Concatena dos listas

**APPEND**

```
L = list()
#L.APPEND(item)
L.append('A')
```

**INSERT**

```
L = list()
#L.INSERT(idx,item)
L.insert(2,'A')
```

**EXTENDS**

```
L = list()
#L.EXTENDS([items])
L.append('A')
```

# List (Lista): AGREGAR elementos a una lista con APPEND

```python

def crearListas():
    # Creacion de listas
    lista1 = []
    lista2 = [1, 2, 3]
    lista3 = [1, "Hello", 3.4]
    lista4 = ["mouse", [8, 4, 6], ['a']]
    lista5 = list()

    #Agregando elementos a la lista
    lista1.append('Hola')
    lista2.append(4)
    lista3.append('@')
    lista4.append([0,1])
    lista5.append(69)

    #imprimir
    print(lista1)
    print(lista2)
    print(lista3)
    print(lista4)
    print(lista5)
#
crearListas()
```

# List (Lista): AGREGAR elementos a una lista con INSERT

```python
def crearListas():
    # Creacion de listas
    lista1 = []
    lista2 = [0, 4, 6]
    lista3 = ["hola", "peru",".","com"]
    lista4 = ["mouse", [8, 6, 4], ['a']]
    lista5 = list()

    #Agregando elementos a la lista
    lista1.insert(0,'Hola')
    lista2.insert(1,2)
    lista3.insert(1,'@')
    lista4.insert(2,[2,0])
    lista5.insert(0,69)

    #imprimir
    print(lista1)
    print(lista2)
    print(lista3)
    print(lista4)
    print(lista5)
#
crearListas()
```

# List (Lista): AGREGAR elementos a una lista con EXTEND

```
1
2  def crearListas():
3      # Creacion de listas
4      lista1 = []
5      lista2 = [10, 11, 12]
6      lista3 = ["hola@", "peru"]
7      lista4 = ["pares", [8, 6, 4]]
8      lista5 = list()
9
10     #Agregando elementos a la lista
11     lista1.extend(['a','e','o'])
12     lista2.extend([13,14])
13     lista3.extend([".","com"])
14     lista4.extend([2])
15     lista5.extend([10,20])
16
17     #imprimir
18     print(lista1)
19     print(lista2)
20     print(lista3)
21     print(lista4)
22     print(lista5)
23 #
24 crearListas()
```

# List (Lista): BUSCAR elementos a una lista

```
1
2  def crearListas():
3      # Creacion de listas
4      lista1 = []
5      lista2 = [10, 11, 12]
6      lista3 = ["hola","@" "peru",".","com"]
7      lista4 = ["pares", [8, 6, 4]]
8      lista5 = list(lista3)
9
10     #Buscando elementos en la lista
11
12     #print(lista1.index())
13     print(lista2.index(10))
14     #print(lista3.index("peru"))
15     #print(lista4.index('x'))
16     print(lista5.index("."))
17 #
18 crearListas()
```

# List (Lista): ELIMINAR elementos a una lista

- FORMAS
  - **REMOVE**: Elimina UNICAMENTE la primera ocurrencia de un valor dentro de la lista.
  - **POP:** Elimina el ULTIMA elemento de la lista y retorna el valor eliminado.

```
REMOVE
L = list([1,2,3])
#L.REMOVE(item)
L.remove(1)
print(L)

resultado
[2,3]
```

```
POP
L = list(["a", "b", "c"])
#L.POP()
u = L.pop()
print(u)

resultado
["c"]
```

# List (Lista): ELIMINAR elementos a una lista con REMOVE()

```
1
2  def crearListas():
3      # Creacion de listas
4      lista1 = []
5      lista2 = [10, 11, 12]
6      lista3 = ["hola","@","peru",".","com"]
7      lista4 = ["pares", [8, 6, 4]]
8      lista5 = list(lista3)
9
10     #Buscando elementos en la lista
11     #lista1.remove(0) #Error lista vacia
12     print(lista1)
13
14     lista2.remove(10)
15     print(lista2)
16
17     lista3.remove("peru")
18     print(lista3)
19
20     lista4.remove([8,6,4])
21     print(lista4)
22
23     lista5.remove(".")
24     print(lista5)
25 #
26 crearListas()
```

# List (Lista): ELIMINAR elementos a una lista con POP()

```python

def crearListas():
    # Creacion de listas
    lista1 = []
    lista2 = [10, 11, 12]
    lista3 = ["hola","@","peru",".","com"]
    lista4 = ["pares", [8, 6, 4]]
    lista5 = list(['a','e','o'])

    #Eliminando elementos en la lista
    #lista1.pop() #Error lista vacia

    print(lista2.pop())
    print(lista2)

    print(lista3.pop())
    print(lista3)

    print(lista4.pop())
    print(lista4)

    print(lista5.pop())
    print(lista5)
#
crearListas()
```

# List (Lista): OPERADORES de lista

- TIPOS DE OPERADOR
  - Operador(+):
    - **Retorna** una **NUEVA lista** como un valor .
    - Es **similar a EXTENDS** (extends agrega una lista en la lista actual).
    - Trabaja en listas, cadenas, enteros y clases definidas por el usuario.
  - Operador(*):
    - Trabaja como un **repetidor de listas**.

# List (Lista): OPERADORES de lista - Ejemplos

```
1
2  def crearlistas():
3      # Creacion de listas
4      lista1 = []
5      lista2 = [10, 11, 12]
6      lista3 = list(['a','e','i'])
7      lista4 = ['o','u']
8
9      # Operador: +
10     lista12 = lista1 + lista2
11     lista34 = lista3 + lista4
12
13     print(lista12)
14     print(lista34)
15
16     # Operador: *
17     print(lista12)*2)
18     print(lista34)*3)
19
20 #
21 crearlistas()
```

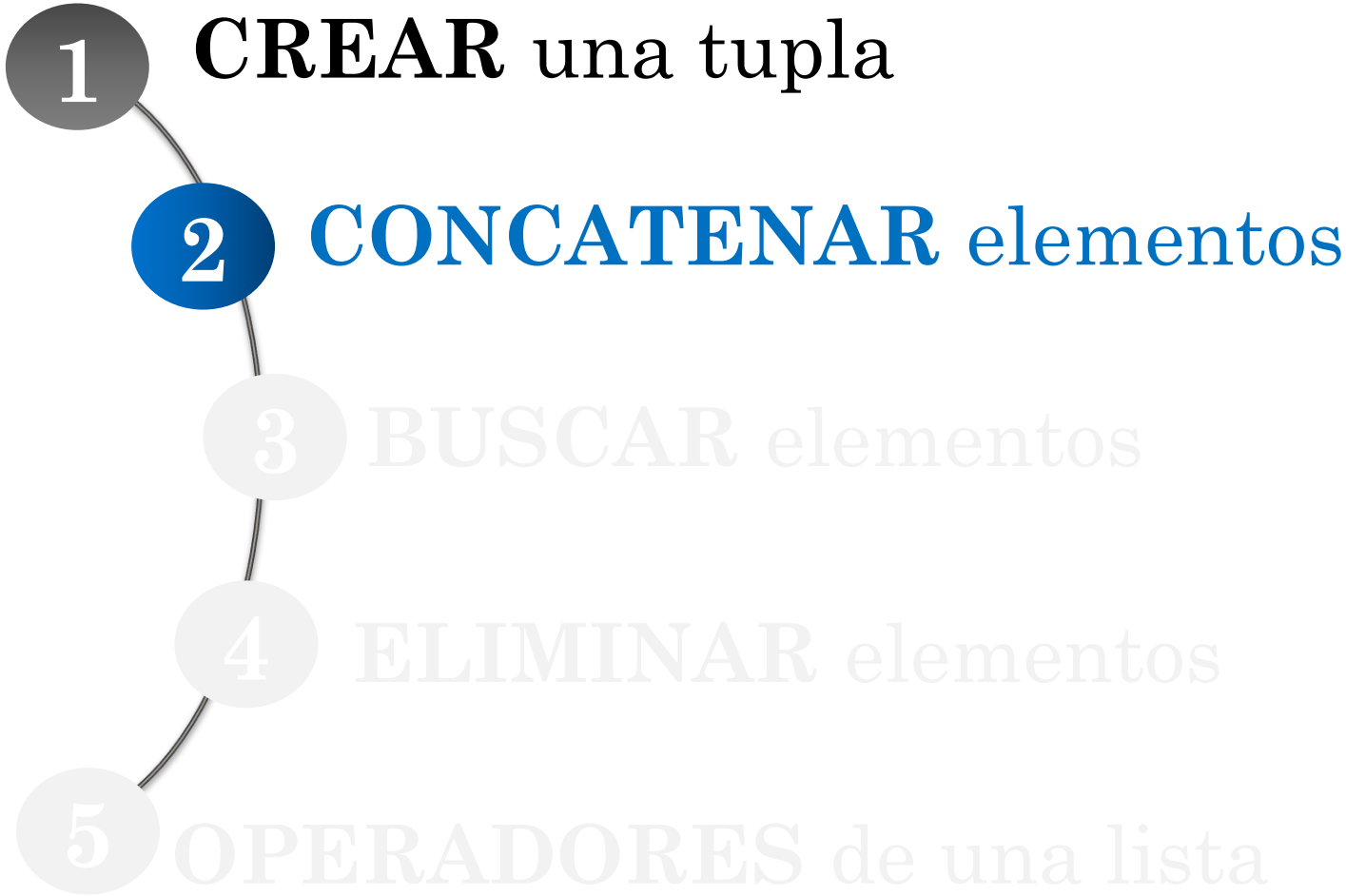
# Tupla

# Tuple (Tuplas) : Conceptos

- En Python una **TUPLA** es:
  - Una lista **INMUTABLE**. No puede cambiar su valor una vez que ha sido creado. Los valores que se da al inicio son los que tendrá durante el ciclo de vida del programa.
  - **Cada valor esta numerado empezando desde cero**.

```
( "Norte" , "Sur" , "Este" , "Oeste" )
  0         1       2          3
```

# Tuple (Tupla): Operaciones sobre una Tupla



# Tuple (Tupla): CREACION de una Tupla

- Una Tupla se crea:
  - Colocando todos los items (elementos) dentro de **PARENTESIS ( ),** separados por coma.
  - Asignando directamente una secuencia de elementos separado por comas a una variable

```
( )
( "0056532',"Lisa","Wong" ,"ININ"  )

tupla1 = ‘norte’, ‘sur’, ‘este’, ’oeste’
```

# Tuple (Tupla): CONCATENAR elementos

- Formas :
  - **Operador(+)**: concatena de tuplas
  - Usar las tuplas a concatenar como elementos de una tupla

```
Operador +
t1 = (‘a’,’e’,’i’)
t2 = «o» , «u»
#concatena dos tuplas
t3 = t1 + t2
```

```
Tuplas como elementos
t1 = (‘a’,’e’,’i’)
t2 = «o» , «u»
#concatena dos tuplas
T3 =(t1,t2)
```

# Dictionary (Diccionario)

# Dictionary (Diccionario) : Conceptos

- En Python una **DICCIONARIO** es:
  - Una colección **NO ORDENADA** de ítems.
  - Una estructura por pares *de la forma* ***key*:value**. La **CLAVE(Key) es de valor único e irrepetible** y un **VALOR**.
  - Un tipo de dato **optimizado** para **recuperar valores** cuando la **clave** es **conocida**.

```
{ 'nombre' : 'Ana' ,'edad' : 19 }
```

clave valor

# Dictionary (Diccionario): Operaciones

# Dictionary (Diccionario): CREAR

- Una Diccionario se crea:
  - Colocando todos los pares de ítems ***key:value*** dentro de **llaves { }**.
  - Utilizando la función dict().
  - Asignando directamente una secuencia de elementos separado por comas a una variable,

```
x = {}
x = {'Ana':25,'Jim':19,'Luisa':24}
x = dict()
```

# Dictionary (Diccionario): ACCESAR elementos

- Formas:
  1. Usando el índice ➔ **[“key”]**.
  2. Usando el método get➔ **get(“key”).**

```
#Ejemplo
vocales = {1:‘a’,2:‘e’,3:‘i’,4:‘o’,5:‘u’}
print(vocales[‘1’]) #usando el indice
print(vocales.get(‘1’)) #usando el método get()
```

# Dictionary (Diccionario): MODIFICAR elementos

- Los diccionario son mutables.
- Se puede **agregar** nuevos itemes o **modificar** el valor de itemes existentes utilizando el **operador de asignación**.
  1. Si la ***clave existe***. El **valor** se **actualiza**.
  2. Si la ***clave no existe***. Se agregar el **nuevo par clave:valor**.

```
#Ejemplo
nums = {'uno':'alfa', 'dos':'omega'}
nums['dos'] = 'beta' # ESTE VALOR SE ACTUALIZA
nums['tres'] = 'gamma' # ESTE VALOR SE AGREGA
print(nums) #valores: 'alfa', 'beta' y 'gama'
```

# Dictionary(Diccionario): ELIMINAR elementos

- FORMAS:
  - **pop(*key*)**: ***Elimina y retorna*** el valor especificado por la clave(***key***).
  - **del(*key*): Elimina el ítem indicado por la clave** (***key***).
  - **popitem(): Elimina y retorna** un ítem arbitrario.
  - **clear(): Elimina todos los ítems** del diccionario.

# Dictionary(Diccionario): ELIMINAR elementos

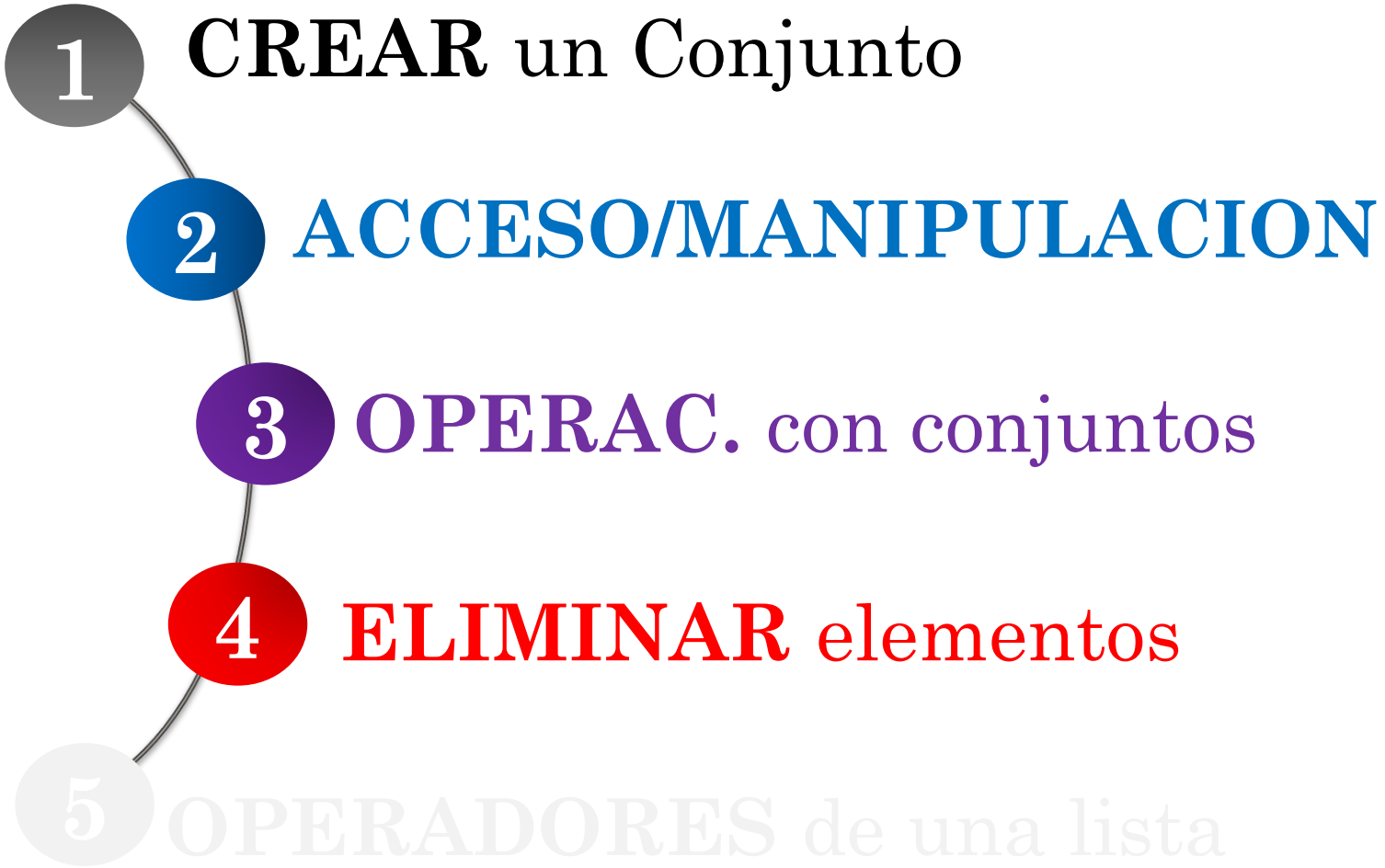
```
def main():
    #crea un diccionario
    cuadrados = {1:1, 2:4, 3:9, 4:16, 5:25}

    # elimina un item con pop(4)  Retorna: 16
    print(cuadrados.pop(4))

    # Nuevo diccionario: {1: 1, 2: 4, 3: 9, 5: 25}
    print(cuadrados)

    # Elimina un item con popitem() y retorna: (1, 1)
    print(cuadrados.popitem())

    # Nuevo diccionario: {2: 4, 3: 9, 5: 25}
    print(cuadrados)

    # elimina un item con del
    del cuadrados[5]

    # Nuevo diccionario
    print(cuadrados)

    # remove all items
    cuadrados.clear()

    # Nuevo diccionario: {}
    print(cuadrados)

main()
```

# Set(Conjunto)

# SET (Conjunto) : Conceptos

- En Python un **SET** es:
  - Una colección de elementos. **NO PERMITE** repetidos.
  - Una estructura que **NO ADMITE** elementos **MUTABLES** (modificación)
  - Puede contener cualquier numero de elementos y pueden ser diferentes tipos (enteros, flotantes, tuplas, cadenas, etc.).

# SET (Conjunto): Operaciones



# SET (Conjunto): CREAR

- Un Conjunto se crea:
  - Utilizando la función set().
  - Colocando los elementos dentro de **llaves { }**.
  - A partir de una Lista
  - A partir de  una cadena

```
s1 = set()
s2 = {1, 2, 3}
s3 = set([1,3,5])
s34= set("DABALE EL ABAD")
```

# SET (Conjunto): ACCESO Y MANIPULACION

- Formas para manipular los elementos:
  1. Agregar ➔ **add(e)**.
  2. Remover ➔ **remove(e).**
  3. Determinar si esta o no esta ➔ **IN, NOT IN**
  4. Longitud del conjunto ➔ **LEN**
  5. **Máximo / Mínimo ➔ MAX, MIN**
  6. **Suma de elementos ➔ SUM**

# SET (Conjunto): ACCESO Y MANIPULACION

```
def main():
    s1 = set([1,2,4])
    s1.add(8)
    s1.add(6)
    s1.remove(8)
    print(s1)
    print("Longitud s1=> " + str(len(s1)))
    print("Maximo en s1=> " + str(max(s1)))
    print("Minimo en s1=> " + str(min(s1)))
    print("Suma en s1=> " + str(sum(s1)))

main()
```

# SET (Conjuntos): OPERACIONES en conjuntos

- En Python las operaciones sobre conjuntos son:
  - **UNION** ➔ Devuelve un nuevo conjunto sumando dos conjuntos.
    - Operador Barra: **|** (barra vertical)
    - Función: **s1.union(s2)**
  - **INTERSECCION** ➔ Devuelve los elementos comunes a dos conjuntos.
    - Operador Andpersand: **&**
    - Función: **s1.intersection(s2)**

# SET (Conjuntos): OPERACIONES en conjuntos

- En Python las operaciones sobre conjuntos son:
  - **DIFERENCIA:** Devuelve los elementos del primero conjunto que no están en el segundo,
    - Operador Diferencia: **-**
    - Función: **s1.difference(s2)**
  - **DIFERENCIA SIMETRICA (Or Exclusivo):** Devuelve los elementos que están en cualquiera de dos conjuntos, pero no en ambos.
    - Operador : **^** (andpersand)
    - Función: **s1.symetric_difference(s2)**

# SET (Conjuntos): OPERACIONES
## Ejemplo

```
def main():
    s1 = {1,2,4}
    s2 = {1,3,5}
    su = s1.union(s2)
    si = s1.intersection(s2)
    sd = s1 - s2
    sds = s1^s2
    print(su)
    print(si)
    print(sd)
    print(sds)

main()
```

# SET(Conjunto): ELIMINAR elementos

- FORMAS
  - **clear():**Elimina todos los elementos del conjunto.
  - **discard():** Remueve un elemento del conjunto.
  - **remove(elemento):** Remueve un elemento del conjunto, lanza  un error si el elemento no pertenece al conjunto.
  - **pop(): remueve y retorna** un elemento arbitrario del conjunto.

# SET(Conjunto): ELIMINAR elementos - Ejemplo

```
def main():
    s1 = {1,2,4}
    s2 = {1,3,5}
    su = s1.union(s2)
    si = s1.intersection(s2)
    sd = s1 - s2
    sds = s1^s2
    print(su)
    print(si)
    print(sd)
    print(sds)
    #Operaciones de eliminacion
    sds.remove(5)
    sds.discard(2)
    print(sds.pop())
    print(sds)
    sds.clear()
    print(sds)

main()
```

# Ejemplos

# Ejemplo 1

**Amigos.SAC es una empresa que está rifando un horno microondas entre sus clientes. Cuando un cliente realiza una compra debe registrar sus datos para el sorteo. Los datos a registrar son: Numero teléfono y nombre. El gerente conocedor de sus habilidades de programación le ha solicitado cargar toda la información en un programa en Python para buscar el nombre del ganador. ¿Cómo lo implementaría?**

**Ejemplo:**

**Numero del cliente: 123456**

**Nombre => Juan Díaz**

```
def main():
    clientes={123456:'Juan Diaz',
              234569:'Ana Yepez',
              147258:'Robert Gonzalez'}
    print("Ingrese Numero a buscar")
    telefono = int(input())
    nombre = clientes.get(telefono)
    print("Nomobre de cliente=> " + str(nombre))


main()
```

# Ejemplo 2

**Los estudiantes de idiomas, están aprendiendo el idioma Quechua. Muchas de las palabras resultan difíciles de escribir. Por lo que, para ayudar a memorizar, sus amigos le han pedido que elabore un programa en Python a modo de diccionario para buscar rápidamente la palabra equivalente entre español y quechua y viceversa.**

**Ejemplo:**

**Buscar: Pirca**

**Equivalente => Pared**

```
def main():
    quechua={'pirca':'pared',
             'pacha':'tierra',
             'huasi':'casa'}
    espaniol={'pared':'pirca',
             'tierra':'pacha',
             'casa':'wasi'}
    print("Buscar")
    clave = str(input())
    valor = quechua.get(clave)
    print("Equivalente: " + valor)


main()
```


gracias
go raibh maith agat
merci
obrigado
danke
謝謝
bedankt
teşekkür ederim
thank you
dziękuję
спасибо
ευχαριστώ
terima kasih
mochchakkeram
hvala
ngiyabonga
sukriya
sagolun
arigatô
tapadh leat
dakujem
kop khun krap
mauruuru
matondo asante
köszönöm
dhanyavad
rahmet
najis tuke
мерси
barka
kia ora
mamnun
sobodi
tanemirt
bayarlalaa
vinaka
chokrane
rahmat
gratias ago
dankie
dank je
shukriya
gràcies
kam sah hamnida
chnorakaloutioun
blagodaram
didi madloba
taafetai lava
obrigada
djiere dieuf
akun
mahalo
murakoze
tau
gràçie
mèsi
spas
dankon
tenki
diolch
grazzi
takk
sulpáy
tack
merce
xièxie
dèkuji
manana
misaotra
kiitos
welalin
nanni
ありがとう
তোমাকে ধন্যবাদ
aciû
enkosi
paldies
taiku
tugarez
dhanyavadagalu