Introducción a Python

Python

- Lenguaje de programación de alto nivel
- Lenguaje interpretado
- Fácil de implementar y entender.
- Usado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning (ML)

Introducción

- Python es un lenguaje de programación de propósito general.
- Algunas características:
  - Poderoso
  - Flexible
  - Sencillo
  - Fácil de aprender
- Es un lenguaje de alto nivel (todo tipo de estructuras de datos, tanto numéricos como de texto).

Python

- Sintaxis muy clara y legible.
- Fuerte capacidad de introspección.
- Orientación a objetos intuitiva.
- Expresión del código procedimental.
- Altamente modular, soporta paquetes jerárquicos.
- Enfocado en el uso de excepciones para el manejo de errores.
- Tipos de datos dinámicos de muy alto nivel.
- Extensa biblioteca estándar (STL)
- Extensiones y módulos fácilmente escritos en C, C + + (o Java para Jython, o. NET para IronPython).

Historia

- Creado en 1991 por Guido van Rossum en los Países Bajos.
- El nombre es en honor al programa cómico de la televisión británica “Monty Python Flying Circus”.
- Python se desarrolla bajo una licencia de Open source.
- Es de código abierto, multi-plataforma es decir se adecua a diversos paradigmas de programación.
- Python se ha vuelto muy popular, es usado en algunos proyectos como Openstack, Blender, Google App Engine, Django, Jupyter, etc.

Historia


www.Tecsify.com
Top 10 Lenguajes de Programación más utilizados
Este es el ranking de los lenguajes de programación más usados en la actualidad, está basado en los datos del índice Tiobe el cual (hay que aclarar) no señala que lenguaje es mejor, sino, que lenguaje se escribió en mayor cantidad durante el último mes del 2022.
#1 Python:
Es un lenguaje muy versátil ya que tiene múltiples áreas de aplicación como inteligencia artificial, Big Data y desarrollo web, es de código abierto y muy fácil de aprender..
#2 Lenguaje C:
Es uno de los primeros lenguajes y forma la base de otros más actuales como C++, C# o Java. Con él podemos desarrollar tanto aplicaciones como sistemas operativos.
#3 Java:
Es un lenguaje de propósito general y su ámbito de aplicación es súper amplio, es orientado a objetos y se utiliza en gran parta para crear aplicaciones y procesos en múltiples dispositivos.
#4 C++:
Surgió como extensión de C para que pudiese manipular objetos, es muy utilizado en bases de datos, navegadores web, compiladores o videojuegos.
#5 C#:
Es un lenguaje muy versátil, creado por Microsoft, similar a C pero orientado a objetos. Es muy usado en la industria de los juegos, robótica, impresión 3D, iot y desarrollo móvil / web.
#6 Visual Basic:
Es un lenguaje orientado a objetos desarrollado por Microsoft. El uso de Visual Basic agiliza y simplifica la creación de aplicaciones .NET con seguridad de tipos.
#7 JavaScript:
Es uno de los preferidos por ser uno de los más poderosos y flexibles, sirve para todo; aplicaciones web, servidores, aplicaciones móviles, no necesita compilación ninguna
#8 Assembly:
Es un lenguaje de programación de bajo nivel. Consiste en un conjunto de instrucciones básicas para los computadores, microprocesadores y otros circuitos integrados programables
#9 SQL:
Es un lenguaje de programación utilizado para el manejo de información en las bases de datos relacionales, además, permite realizar operaciones de acceso y manipulación de la información.
#10 PHP:
PHP es uno de los clásicos; es un lenguaje de uso general que se adapta especialmente al desarrollo web, PHP se encuentra instalado en más de 20 millones de sitios web.
/Tecsify @Tecsify @Tecsify
Tecsify
www.Tecsify.com/blog
/Tecsify Tecsify @Tecsify

Aplicaciones

- Aplicaciones de escritorio.
- Aplicaciones web.
- Análisis de datos.
- Administración de servidores.
- Seguridad y análisis de penetración.

Aplicaciones

- Cómputo en la nube.
- Cómputo científico.
- Análisis de lenguaje natural.
- Visión artificial.
- Animación, videojuegos e imágenes generadas por computadora.
- Aplicaciones móviles.

Evolución

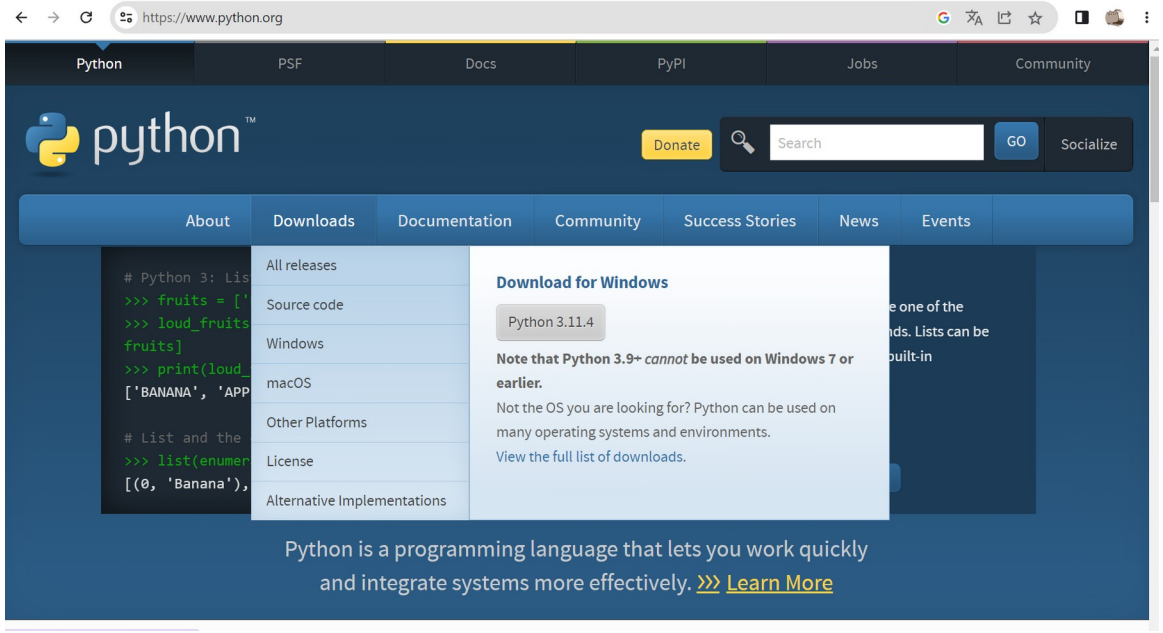
- Actualmente existen dos versiones de Python cuyo código no es compatible.
- Python 3 es una versión revisada del lenguaje, la cual fue publicada en 2009 y que incluye modificaciones y mejoras que lo hacen incompatible con código de versiones previas; mientras que Python 2 es una versión que es compatible con código antiguo.

Evolución

- Python 2.7 es una versión "de transición" que permite usar algunos elementos sintácticos tanto de Python 2 como de Python 3.
- Python 3 presenta mejoras notables con respecto a Python 2, sin embargo aún existe mucho código legado compatible con Python 2.
- Se prevé que la publicación y soporte de nuevas versiones de Python 2 cese a partir del año 2020.

Instalación

Página: https://www.python.org/



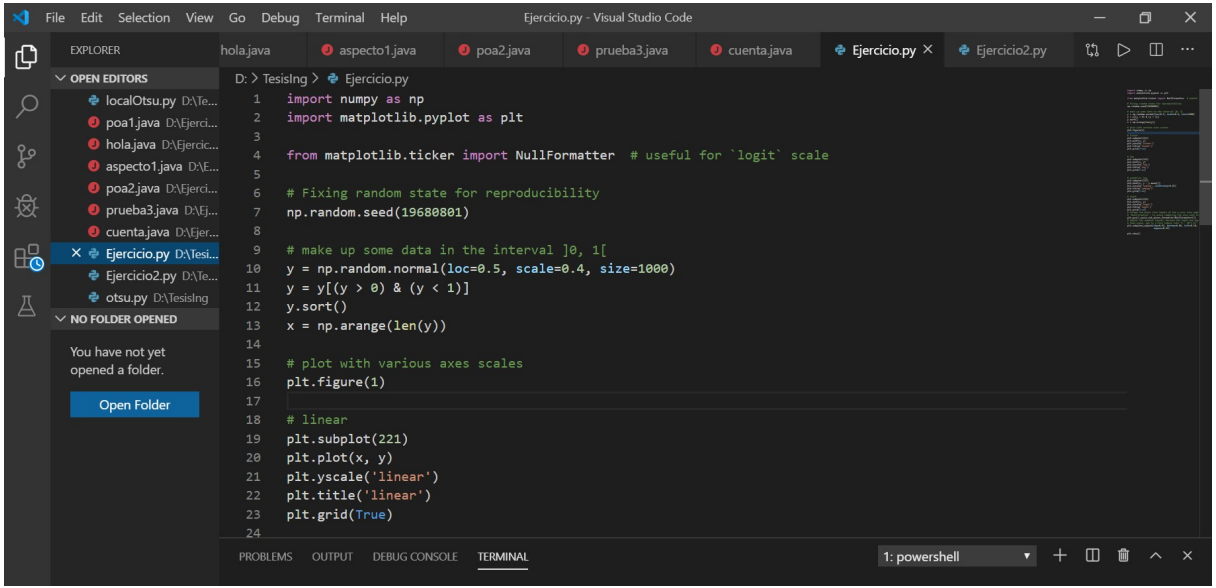
Instalación

Link de ayuda:

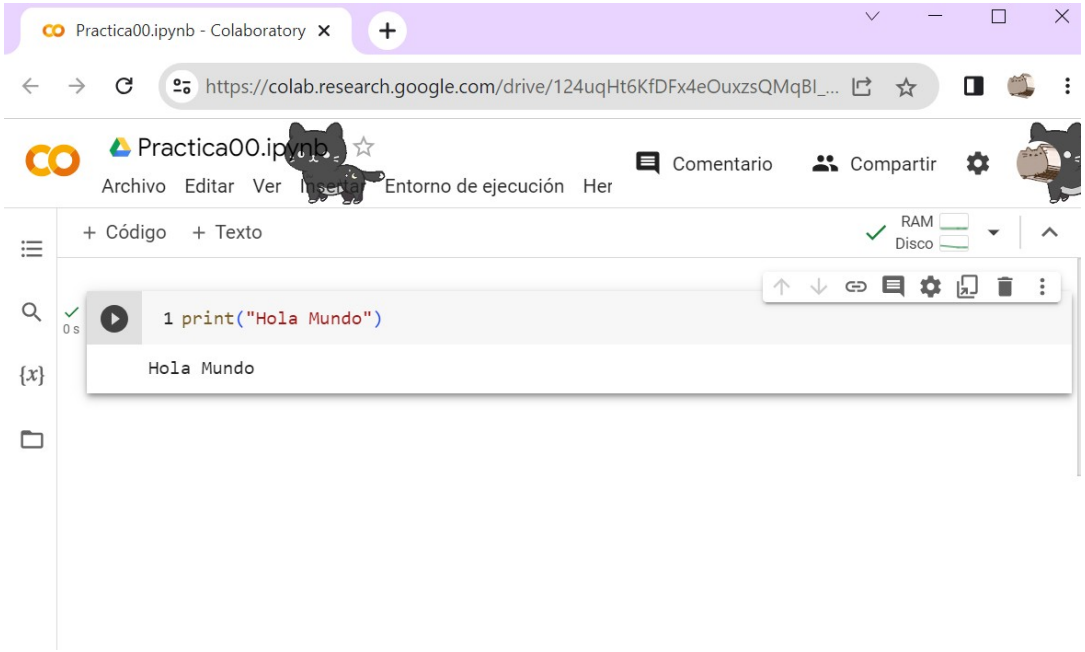
https://python-para-impacientes.blogspot.com/2017/02/instalar-python-paso-paso.html

Editores

- Pycharm
- Visual Studio Code
- PyDev
- PyScripter



Editores


Practica00.ipynb - Colaboratory
https://colab.research.google.com/drive/124uqHt6KfDFx4eOuxzsQMqBI_...
Practica00.ipynb
Archivo Editar Ver Entorno de ejecución Her
Comentario
Compartir
+ Código + Texto
RAM
Disco
1 print("Hola Mundo")
0 s
Hola Mundo

Google Colab

- Colaboratory, o "Colab"
- Permite a cualquier usuario escribir y ejecutar código arbitrario de Python en el navegador.
- Es especialmente adecuado para tareas de aprendizaje automático, análisis de datos y educación.
- Desde un punto de vista más técnico, Colab es un servicio de cuaderno alojado de Jupyter que no requiere configuración y que ofrece acceso sin coste adicional a recursos informáticos, como GPUs.

Primer programa en Python

- Hello world!

Contenido del módulo

- Introducción a Python
- **Variables y operaciones**
- Listas y cadenas
- Estructuras del control
- Funciones y módulos

Variables, Operadores, Expresiones y Sentencias

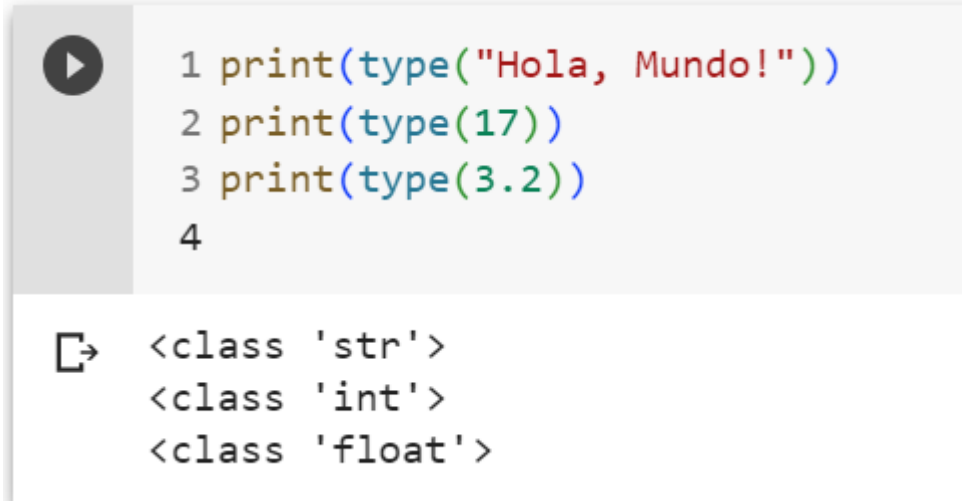
Valores y tipos

- Un valor es una de los cosas fundamentales como una letra o un número que un programa manipula.
- Los valores que hemos visto hasta ahora son 2 (el resultado cuando añadimos 1 + 1), y "Hola todo el Mundo!".

  - 1+1 = 2 → ENTERO.
  - “Hola todo el mundo” → CADENA

Valores y tipos

- Para revisar que tipo tiene un valor se usa la función **type( )**.



```
1 print(type("Hola, Mundo!"))
2 print(type(17))
3 print(type(3.2))
4
```

```
<class 'str'>
<class 'int'>
<class 'float'>
```

Valores y tipos - Observaciones

- Se está usando la función **print( )** para mostrar información por pantalla.
  - print(variable)
  - print(valor)

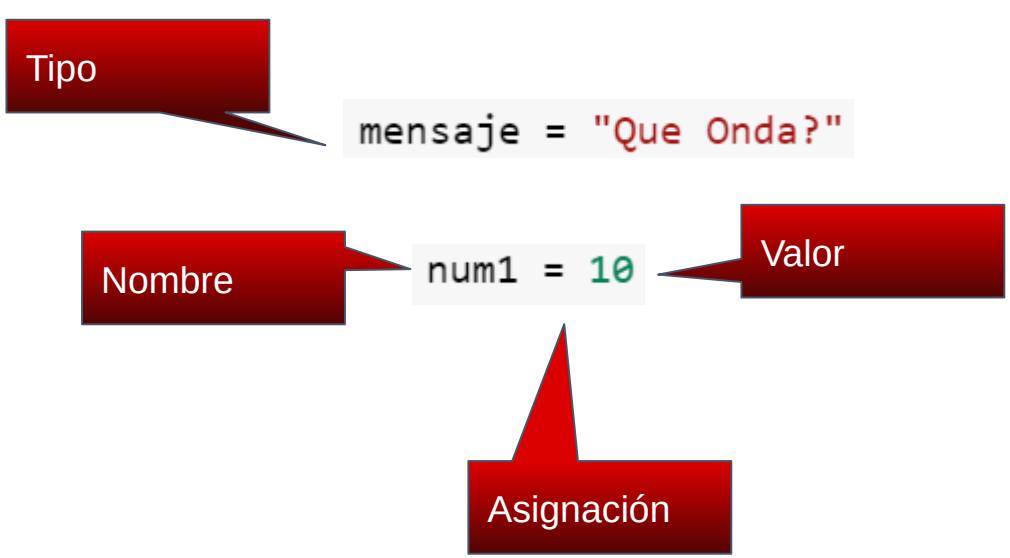
- Los tipos de **datos básicos** de Python son los booleanos, los numéricos (enteros, punto flotante y complejos) y las cadenas de caracteres.

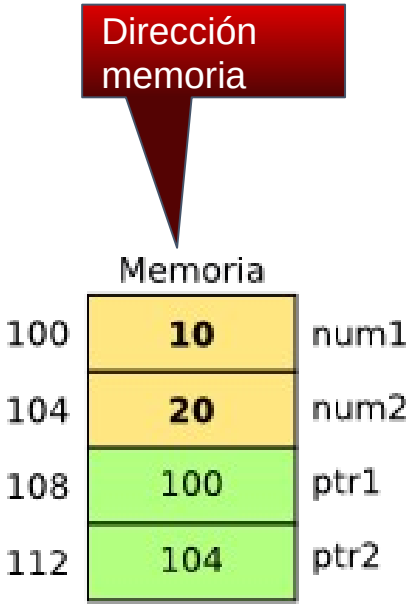
Variables

- Una de las características más poderosas en un lenguaje de programación es la capacidad de **manipular variables**.
- Las variables permiten **asignar nombres** coherentes a información para ser reutilizada con facilidad.
- Una variable es un **nombre** que se refiere a un valor.
- La **sentencia de asignación** crea nuevas variables y les da valores:

```
1 mensaje = "Que Onda?"
2 n = 17
3 pi = 3.14159
```

Variables


Tipo
mensaje = "Que Onda?"
Nombre
num1 = 10
Valor
Asignación


Dirección memoria
Memoria
100
10
num1
104
20
num2
108
100
ptr1
112
104
ptr2

Variables

- La función **print( )** también funciona con variables.
- En cada caso el resultado es el valor de la variable.

```
1 mensaje = "Que Onda?"
2 num1 = 10
3 pi = 3.14159
4 print(mensaje)
5 print(num1)
6 print(pi)
```

```
Que Onda?
10
3.14159
```

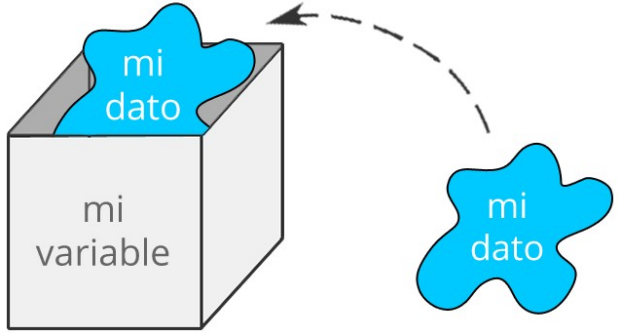
Variables

- Las variables tienen tipos.
- El **tipo de una variable** es el **tipo del valor** al que se refiere.

```
1 mensaje = "Que Onda?"
2 num1 = 10
3 pi = 3.14159
4 print(type(mensaje))
5 print(type(num1))
6 print(type(pi))
7
```

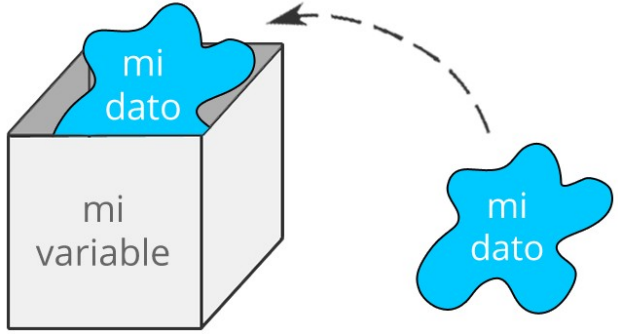
```
<class 'str'>
<class 'int'>
<class 'float'>
```

Variables



- **Nombres de las variables**
  - Escoger nombres significativos para sus variables , que se usa la variable.
  - Los nombres pueden ser arbitrariamente largos, teniendo en consideración:
    - Tienen que empezar por una letra o guión bajo.
    - Pueden ser compuestos, separado por guión bajo.
    - Pueden estar en minúsculas o mayúscula (CASE SENSITIVE)
    - El uso de **keywords (palabras reservadas)** como nombres está prohibido.

Variables

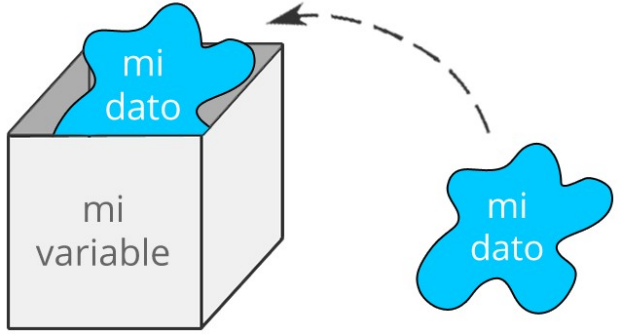


- **Nombres de las variables**
  - Tienen que empezar por una letra o guión bajo.
  - Pueden ser compuestos, separado por guión bajo.

```
1 cat_color = 'Brown'
2 cat_Color = 'White'
3 _threads = 8
4 variable = 10
5 2phone_number = 78469334212
6
```

```
  File "<ipython-input-12-b4b0e9d3fdd3>", line 5
    2phone_number = 78469334212
    ^
SyntaxError: invalid decimal literal
```

Variables

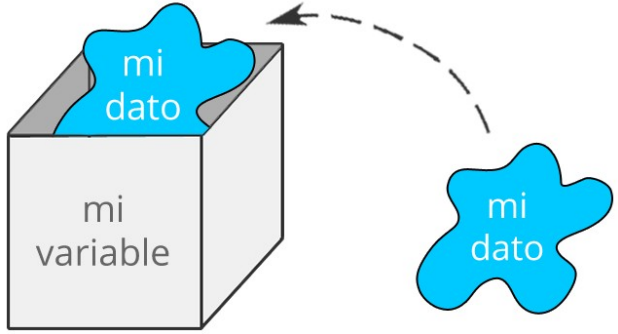


- **Nombres de las variables**
  - Pueden estar en minúsculas o mayúscula
    Recuerde: la capitalización importa, por lo cual, **Pedro** y **pedro** son variables diferentes.

```
1 cat_color = 'Brown'
2 cat_Color = 'White'
3 print(cat_Color)
4 print(cat_color)
```

```
White
Brown
```

Variables



- **Nombres de las variables**
  - El uso de **keywords (palabras reservadas)** como nombres está prohibido.
    Palabras reservadas: definen las reglas del lenguaje y su estructura, y no pueden ser usadas como nombres de variables. Python tiene un poco más de treinta palabras reservadas:

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

Operadores

Los operadores son símbolos que le indican al intérprete que realice una operación específica, como aritmética, comparación, lógica, etc.
Estos son los diferentes tipos de operadores en Python:

- Operadores aritméticos
- Operadores relacionales
- Operadores de asignación
- Operadores lógicos

Operadores y operandos

- Los Operadores son símbolos especiales que representan cómputos como la suma y la multiplicación.
- Los valores que el operador usa se denominan **operandos**.
- Las siguientes son expresiones válidas en Python:

```
20+32
hora-1
hora*60+minuto
minuto/60
5**2
(5+9)*(15-7)
```

Operadores aritméticos

OPERADOR	DESCRIPCIÓN	USO
+	Realiza Adición entre los operandos	12 + 3 = 15
-	Realiza Sustracción entre los operandos	12 - 3 = 9
*	Realiza Multiplicación entre los operandos	12 * 3 = 36
/	Realiza División entre los operandos	12 / 3 = 4
%	Realiza un módulo entre los operandos	16 % 3 = 1
**	Realiza la potencia de los operandos	12 ** 3 = 1728
//	Realiza la división con resultado de número entero	18 // 5 = 3

```
1 print(12 + 3)
2 print(12 - 3)
3 print(12 * 3)
4 print(12 / 3)
5 print(16 % 3)
6 print(12 ** 3)
7 print(18 // 5)
8 print(18 / 5)
```

```
15
9
36
4.0
1
1728
3
3.6
```

Operadores aritméticos

- Cuando hay más de un operador en una expresión, el orden de evaluación depende de las reglas de precedencia.
  - Python sigue las mismas reglas de precedencia a las que estamos acostumbrados para sus operadores matemáticos.
  - El acrónimo **PEMDAS** es útil para recordar el orden de las operaciones:
    - Paréntesis
    - Exponenciación
    - Multiplicación / División
    - Adición / Sustracción

Operadores relacionales

OPERADOR	DESCRIPCIÓN	USO
>	Devuelve True si el operador de la izquierda es mayor que el operador de la derecha	12 > 3 devuelve True
<	Devuelve True si el operador de la derecha es mayor que el operador de la izquierda	12 < 3 devuelve False
==	Devuelve True si ambos operandos son iguales	12 == 3 devuelve False
>=	Devuelve True si el operador de la izquierda es mayor o igual que el operador de la derecha	12 >= 3 devuelve True
<=	Devuelve True si el operador de la derecha es mayor o igual que el operador de la izquierda	12 <= 3 devuelve False
!=	Devuelve True si ambos operandos no son iguales	12 != 3 devuelve True

```
1 print(12 > 3)
2 print(12 < 3)
3 print(12 == 3)
4 print(12 >= 3)
5 print(12 <= 3)
6 print(12 != 3)
7
```

```
True
False
False
True
False
True
```

Operadores de asignación

OPERADOR	DESCRIPCIÓN	USO
=	a = 5. El valor 5 es asignado a la variable a	=
+=	a += 5 es equivalente a a = a + 5	+=
-=	a -= 5 es equivalente a a = a - 5	-=
*=	a *= 3 es equivalente a a = a * 3	*=
/=	a /= 3 es equivalente a a = a / 3	/=

```
1 a = 5
2 print(a)
3 a += 5
4 print(a)
5 a -= 5
6 print(a)
7 a *= 3
8 print(a)
9 a /= 3
10 print(a)
11
```

```
5
10
5
15
5.0
```

Operadores lógicos

OPERADOR	DESCRIPCIÓN	USO
and	Devuelve True si ambos operandos son True	a and b
or	Devuelve True si alguno de los operandos es True	a or b
not	Devuelve True si alguno de los operandos False	not a

Operaciones sobre cadenas

En general, no se puede calcular operaciones matemáticas sobre cadenas, incluso si las cadenas lucen como números.

Las siguientes operaciones son inválidas (asumiendo que mensaje tiene el tipo cadena):

- mensaje = “Soy de Peru”
- mensaje-1
- "Hola"/123
- mensaje*"Hola"
- "15"+2

```
1 mensaje = "Soy de Peru"
2 mensaje-1
3 "Hola"/123
4 mensaje*"Hola"
5 "15"+2
6
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-20-1466dca96904> in <cell line: 2>()
      1 mensaje = "Soy de Peru"
----> 2 mensaje-1
      3 "Hola"/123
      4 mensaje*"Hola"
      5 "15"+2

TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

Operaciones sobre cadenas

- El **operador +** funciona con cadenas, aunque no calcula lo que esperamos. Representa la concatenación, que significa unir los dos operandos enlazándolos en el orden en que aparecen.

```
1 msj1 = "Hola"
2 msj2 = "Amigo"
3 print(msj1+msj2)
```

```
HolaAmigo
```

- El **operador** * funciona con cadenas, concatenando varias veces una de ellas.

```
1 msj1 = "Hola"
2 print(msj1*5)
```

```
HolaHolaHolaHolaHola
```

Operaciones sobre cadenas

Las letras mayúsculas vienen antes que las minúsculas es decir son menores que las minúsculas.
Para ordenarlas, se usa el orden ortográfico.

```
1 print("hola" <= "ola")
2 print("hola" <  "ola")
3 print("hola" >= "ola")
4 print("hola" >  "ola")
5 print("hola" == "ola")
6 print("hola" == "HOLA")
7 print("hola" >  "HOLA")
8
```

```
True
True
False
False
False
False
True
```

Expresiones

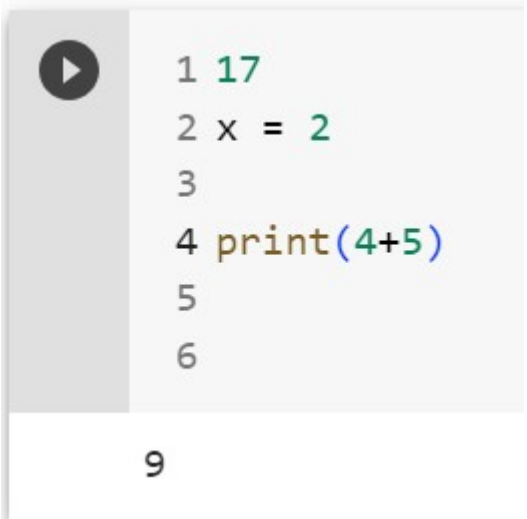
- Una **expresión** es una combinación de valores, variables y operadores.
  - 1 + 3
  - valor1 + 5
  - (valor1 * 5) + radio
  - 1 3 * - (no es aceptado porque no sigue las reglas de sintaxis)
- La expresiones son **evaluadas** por el intérprete
  - Si usted digita una expresión en la línea de comandos, el intérprete la evalúa y despliega su resultado:

```
1 valor1 = 7
2 nro = 9
3 total = (valor1 * 5) + nro
4 print(total)
```

```
44
```

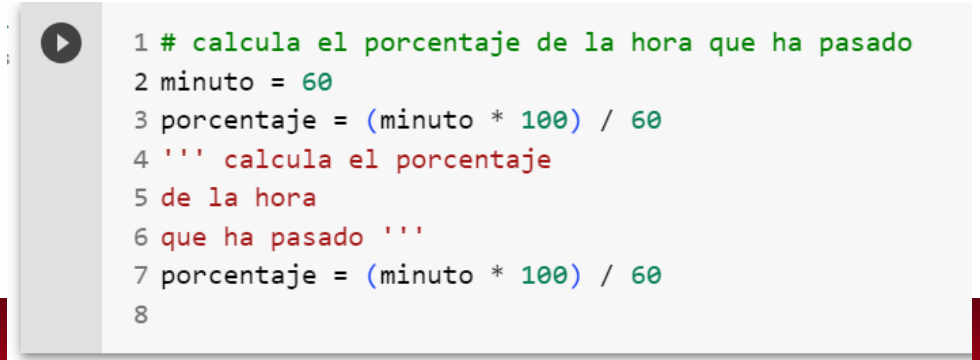
Expresiones

- Un valor, por sí mismo, se considera como una expresión, lo mismo ocurre para las variables.
- Aunque es un poco confuso, evaluar una expresión no es lo mismo que imprimir o desplegar un valor. En el primer caso (líneas 1 y 2) la expresión se evalúa, es decir, se va a calcular la operación, sin embargo este resultado no se va a imprimir. En el segundo caso (línea 4), además de evaluarse la expresión está también se imprime.



Comentarios

- A medida que los programas se hacen más grandes y complejos, se hacen más difíciles de leer. Los lenguajes formales son densos; y a menudo, es difícil mirar una sección de código y saber que hace, o por qué lo hace.
- Por esta razón, es una muy buena idea añadir notas a sus programas para explicar en lenguaje natural lo que el programa hace. Estas notas se denominan **comentarios** y se marcan con el símbolo **#** para cada línea, o entre 3 apóstrofos **(''')** para varias líneas.



```
# calcula el porcentaje de la hora que ha pasado
minuto = 60
porcentaje = (minuto * 100) / 60
''' calcula el porcentaje
de la hora
que ha pasado '''
porcentaje = (minuto * 100) / 60

```

Contenido del módulo

- Introducción a Python
- Variables y operaciones
- **Listas y cadenas**
- Estructuras del control
- Funciones y módulos

Listas y Cadenas

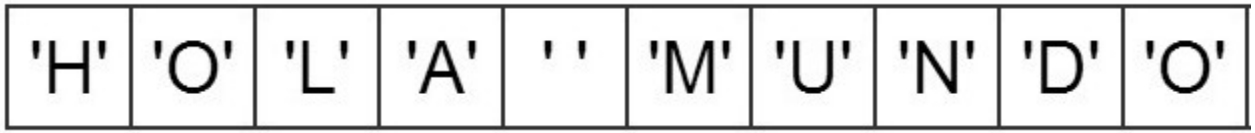
Contenido

- **Cadenas**
  - **Concepto**
  - **Tamaño**
  - **Segmentos**
  - **Funciones**

- Listas
  - Concepto
  - Tamaño
  - Segmentos
  - Modificaciones
  - Anidamiento

Cadenas

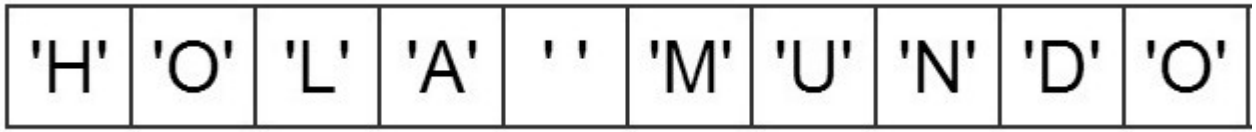
- Las **cadenas (o strings)** son un tipo de datos compuestos por secuencias de caracteres que representan texto.
- Estas cadenas de texto son de tipo str y se delimitan mediante el uso de comillas simples o dobles.




```
cadena ="HOLA MUNDO"
```

Cadenas

- Dependiendo de lo que hagamos podemos tratar un tipo compuesto como unidad o podemos acceder a sus partes.



- Si se desea acceder a un carácter de una cadena se usa el **operador corchete []** y el **índice** deseado.

- Un **índice** especifica un elemento de un conjunto ordenado, en este caso el conjunto de caracteres de la cadena.

Cadenas

Índices:	0	1	2	3	4	5	6	7	8	9
cadena =	'H'	'O'	'L'	'A'	' '	'M'	'U'	'N'	'D'	'O'

```
1 cadena ="HOLA MUNDO"
2 print(cadena)
3 letra =cadena[3]
4 print(letra)
```

```
HOLA MUNDO
A
```

Cadenas

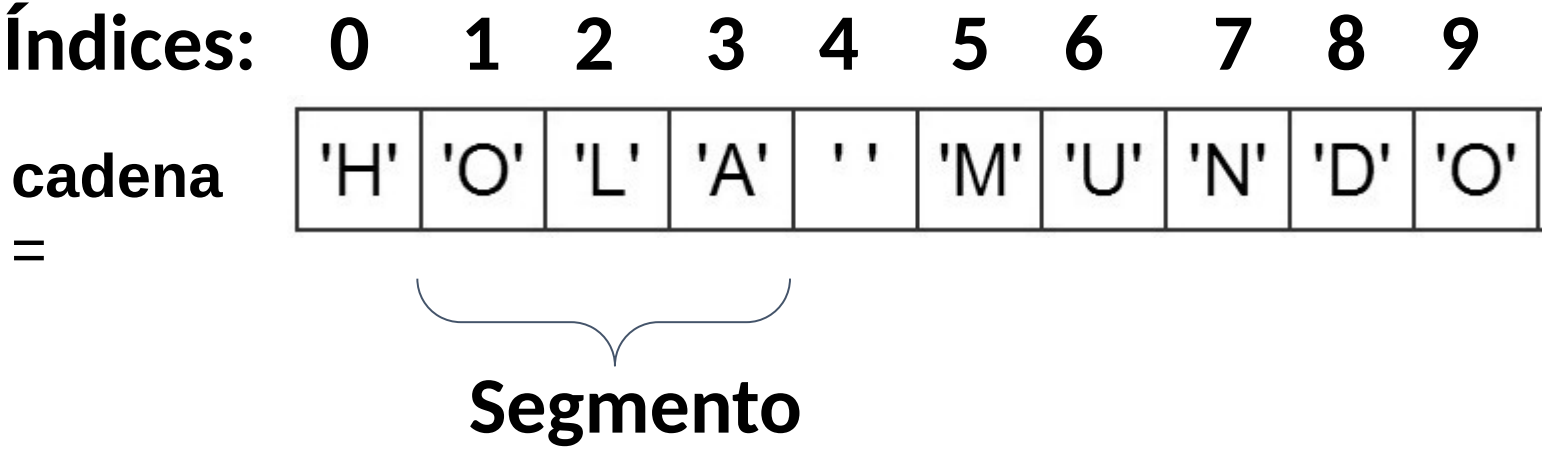
Índices:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
cadena =	'H'	'O'	'L'	'A'	' '	'M'	'U'	'N'	'D'	'O'

```
1 cadena ="HOLA MUNDO"
2 longitud =len(cadena)
3 print(longitud)
4 print(cadena[longitud-1])
5 print(cadena[len(cadena)-1])
```

```
10
O
O
```

Cadenas

- Un **segmento** de una cadena es una porción de una cadena

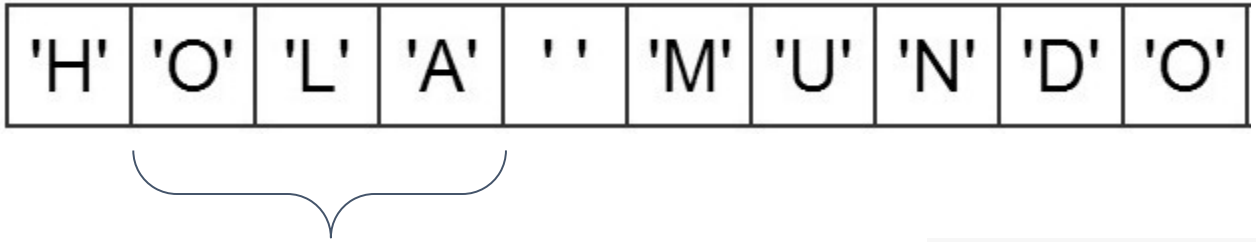


Cadenas

- Para **extraer** un segmento se usa el operador corchete[] con el siguiente formato**[n:m]**, que significa que el segmento extraído va desde del índice n al índice m-1.

Cadenas

Índices: **0 1 2 3 4 5 6 7 8 9**



Segmento [1:4]

```
1 cadena ="HOLA MUNDO"
2 print(cadena[1:4])
```

```
OLA
```

Cadenas

- Las cadenas creadas de la forma explicada hasta ahora, son constantes es decir sus caracteres no pueden cambiar.

```
1 saludo = "Hola Mario"
2 saludo[9] = 'a'
3 print(saludo)
4
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-c1ae4d0eacdc> in <cell line: 2>()
      1 saludo = "Hola Mario"
----> 2 saludo[9] = 'a'
      3 print(saludo)

TypeError: 'str' object does not support item assignment
```

BUSCAR EN STACK OVERFLOW

Cadenas

- **Solución:** Crear una nueva cadena, concatenar y extraer

```
1 saludo = "Hola Mario"
2 print(len(saludo))
3 nuevo_saludo = saludo[0:len(saludo)-1]+'a'
4 print(nuevo_saludo)
```

```
10
Hola Maria
```

Cadenas

- **Invertir cadena**

  nombreCadena[inicio:fin:**paso**]

```
1 cadena = "Hola Python"
2 print(cadena[::-1])
```

```
nohtyP aloH
```

Cadenas

- **Función : find()**

```
1 cadena = "Hola Python"
2 print(cadena.find('P'))
3
```

```
5
```

```
1 print("hola","\n",7)
```

```
hola
 7
```

- **Función : replace()**

```
1 cadena = "Hola Python"
2 print(cadena.replace('P','p'))
```

```
Hola python
```

Listas

- Son **similares a las cadenas**,
- Son una colección de elementos que pueden ser de cualquier tipo (números, palabras, objetos, funciones, etc).

  - Cadena:

    ```
    mi_cadena = "PVJ"
    ```

  - Lista:

    ```
    mi_lista = ["PVJ", 2.0, 5, [2, 5]]
    ```

```
1 mi_cadena = "PVJ"
2 mi_lista = ["PVJ", 2.0, 5, [2, 5]]
```

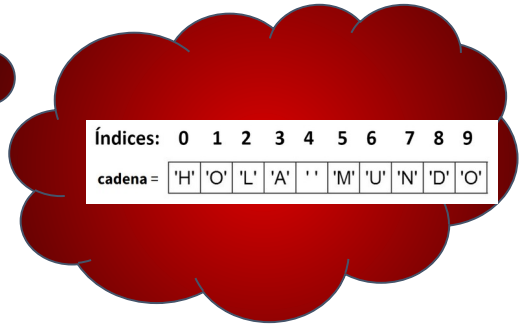
Listas

- Las listas se **definen entre corchetes ([ ])**, y los elementos se separan por comas.

```
1 a = [100, 2, 3, 4, 5]
2 b = ["hola", "como", "estas"]
3 c = ["hola", 2.0, 5, [10, 20]]
4 d = [] #lista vacía
```

Listas

- **Acceder** a sus elementos mediante su índice
  - ***Error:*** Si escribimos mi_lista[2.0]



- Obtener la longitud de una lista es similar a las cadenas, ***len(mi_lista)***

```
cadena ="HOLA MUNDO"
longitud =len(cadena)
```

Listas

- **Concatenación**, al igual que las cadenas se pueden concatenar listas.

```
1 lista1 = [1,4,"Hola"]
2 lista2 = ["PVJ", 2.0, 5]
3 lista3 = lista1 + lista2
4 print(lista3)
```

```
[1, 4, 'Hola', 'PVJ', 2.0, 5]
```

Listas

- **Segmentos:** similar que en las cadenas

```
1 lista = [1, 4, 'Curso', 'PVJ', 2.0, 5]
2 print(len(lista))
3 s1 = lista[0:5]
4 s2 = lista[ :5]
5 s3 = lista[3:]
6 s4 = lista[:]
7 print(s1)
8 print(s2)
9 print(s3)
10 print(s4)
```

```
6
[1, 4, 'Curso', 'PVJ', 2.0]
[1, 4, 'Curso', 'PVJ', 2.0]
['PVJ', 2.0, 5]
[1, 4, 'Curso', 'PVJ', 2.0, 5]
```

Listas

- Las listas a diferencia de las cadenas son **mutables** quiere decir que pueden ser modificadas:
  - modificar
  - eliminar o
  - agregar

  elementos en una lista

Listas

- **Modificar** un elemento

```
1 #Cambiar el texto de Curso por Tema
2 lista = [1, 4, 'Curso', 'PVJ', 2.0, 5]
3 print(lista)
4 lista[2]="Tema"
5 print(lista)
```

```
[1, 4, 'Curso', 'PVJ', 2.0, 5]
[1, 4, 'Tema', 'PVJ', 2.0, 5]
```

Listas

- **Agregar** elementos:
  - ➢Al inicio
  - ➢Al medio
  - ➢Al final.

```
1 lista = [1, 4, 'Curso', 'PVJ', 2.0, 5]
2 print(lista)
3 #Agregar al inicio
4 lista[0:0] = ["Soy Nuevo"]
5 print(lista)
6 #Agregar al medio
7 lista[3:3] = ["Amigo"]
8 print(lista)
9 #Agregar al final
10 lista[8:8] = ["Final"]
11 print(lista)
12 lista.append("Otro Final")
13 print(lista)
```

```
[1, 4, 'Curso', 'PVJ', 2.0, 5]
['Soy Nuevo', 1, 4, 'Curso', 'PVJ', 2.0, 5]
['Soy Nuevo', 1, 4, 'Amigo', 'Curso', 'PVJ', 2.0, 5]
['Soy Nuevo', 1, 4, 'Amigo', 'Curso', 'PVJ', 2.0, 5, 'Final']
['Soy Nuevo', 1, 4, 'Amigo', 'Curso', 'PVJ', 2.0, 5, 'Final', 'Otro Final']
```

Listas

- **Remover** elementos

```
1 lista = [1, 4, 'Curso', 'PVJ', 2.0, 5]
2 print(lista)
3 #Eliminar Curso
4 lista[2:3] =[]
5 print(lista)
6 #Eliminar 2.0
7 lista.remove(2.0)
8 print(lista)
```

```
[1, 4, 'Curso', 'PVJ', 2.0, 5]
[1, 4, 'PVJ', 2.0, 5]
[1, 4, 'PVJ', 5]
```

Listas

```
1 lista = [1, 4, 'Curso', 'PVJ', 2.0, 5]
2 print(lista)
3 #Agregar
4 lista[0:0] = ["Soy Nuevo", 44, True]
5 print(lista)
6 lista[5:5] = ["Aqui", False]
7 print(lista)
8 #Eliminar
9 lista[2:5] =[]
10 print(lista)
```

```
[1, 4, 'Curso', 'PVJ', 2.0, 5]
['Soy Nuevo', 44, True, 1, 4, 'Curso', 'PVJ', 2.0, 5]
['Soy Nuevo', 44, True, 1, 4, 'Aqui', False, 'Curso', 'PVJ', 2.0, 5]
['Soy Nuevo', 44, 'Aqui', False, 'Curso', 'PVJ', 2.0, 5]
```

Listas

Copiar listas

```
1 lista = [1, 4, 'Curso', 'PVJ', 2.0, 5]
2 otraLista = lista
3 print(lista)
4 print(otraLista)
```

```
[1, 4, 'Curso', 'PVJ', 2.0, 5]
[1, 4, 'Curso', 'PVJ', 2.0, 5]
```

Listas

Copiar listas no se realiza con el operador de asignación (=), ya que ….

Se está nombrando una nueva variable que hace referencia a la lista original. Esto se conoce como **ALIAS**

```
1 lista = [1, 4, 'Curso', 'PVJ', 2.0, 5]
2 otraLista = lista
3 print(lista)
4 print(otraLista)
5 lista[1] = 40
6 print(lista)
7 print(otraLista)
8
```

```
[1, 4, 'Curso', 'PVJ', 2.0, 5]
[1, 4, 'Curso', 'PVJ', 2.0, 5]
[1, 40, 'Curso', 'PVJ', 2.0, 5]
[1, 40, 'Curso', 'PVJ', 2.0, 5]
```

Listas

Copiar listas :

- Usando segmentos
- Importando librería

```
1 lista = [1, 4, 'Curso', 'PVJ', 2.0, 5]
2 otraLista = lista[:]
3 print(lista)
4 print(otraLista)
5 lista[1] = 40
6 print(lista)
7 print(otraLista)
```

```
[1, 4, 'Curso', 'PVJ', 2.0, 5]
[1, 4, 'Curso', 'PVJ', 2.0, 5]
[1, 40, 'Curso', 'PVJ', 2.0, 5]
[1, 4, 'Curso', 'PVJ', 2.0, 5]
```

Listas

Copiar listas :

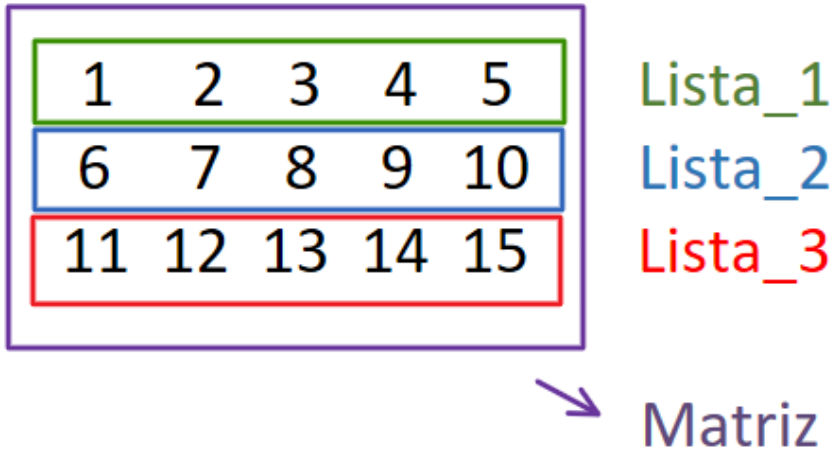
- Usando segmentos
- Importando librería

```
1 import copy
2 lista = [1, 4, 'Curso', 'PVJ', 2.0, 5]
3 otraLista = copy.deepcopy(lista)
4 print(lista)
5 print(otraLista)
6 lista[1] = 40
7 print(lista)
8 print(otraLista)
```

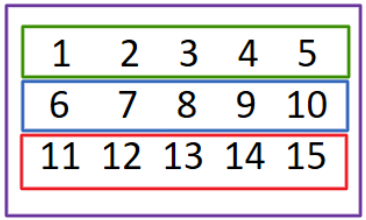
```
[1, 4, 'Curso', 'PVJ', 2.0, 5]
[1, 4, 'Curso', 'PVJ', 2.0, 5]
[1, 40, 'Curso', 'PVJ', 2.0, 5]
[1, 4, 'Curso', 'PVJ', 2.0, 5]
```

Listas Anidadas

- Una lista anidada aparece como elemento dentro de otra lista.
- Las **matrices** se pueden implementar como listas anidadas, es decir, como una lista de listas.



Listas Anidadas



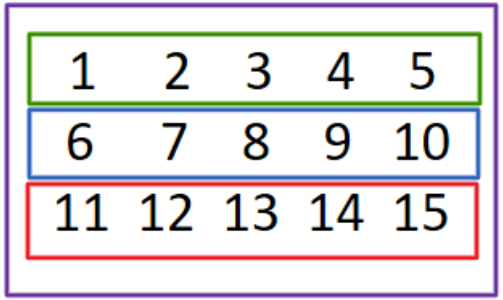

```
1 lista_1 = [1,2,3,4,5]
2 lista_2 = [6,7,8,9,10]
3 lista_3 = [11,12,13,14,15]
4 matriz=[lista_1,lista_2,lista_3]
5 print(matriz)
6 matriz2 = [[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]]
7 print(matriz2)
```

```
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]
```

Listas Anidadas

Se pueden realizar las siguientes operaciones:

- **Acceder** a un dato: nombre_matriz[fila][columna]



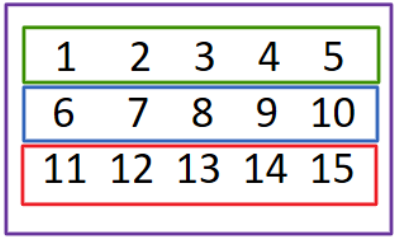

```
1 matriz = [[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]]
2 print(matriz[2][3])
```

```
14
```

Listas Anidadas

Se pueden realizar las siguientes operaciones:

- El tamaño de la matriz




```
1 matriz = [[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]]
2 #Filas
3 print("Filas:",len(matriz))
4 #Columnas
5 print("Columnas:",len(matriz[0]))
```

```
Filas: 3
Columnas: 5
```

Listas Anidadas

Para **copiar listas anidadas** no se puede trabajar con segmentos, sino con la librería **copy**

```
1 import copy
2 matriz = [[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]]
3 otraMatriz =  copy.deepcopy(matriz)
4 print(matriz)
5 print(otraMatriz)
6 matriz[1][1] = 100
7 print(matriz)
8 print(otraMatriz)
```

```
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]
[[1, 2, 3, 4, 5], [6, 100, 8, 9, 10], [11, 12, 13, 14, 15]]
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]
```

Gracias!