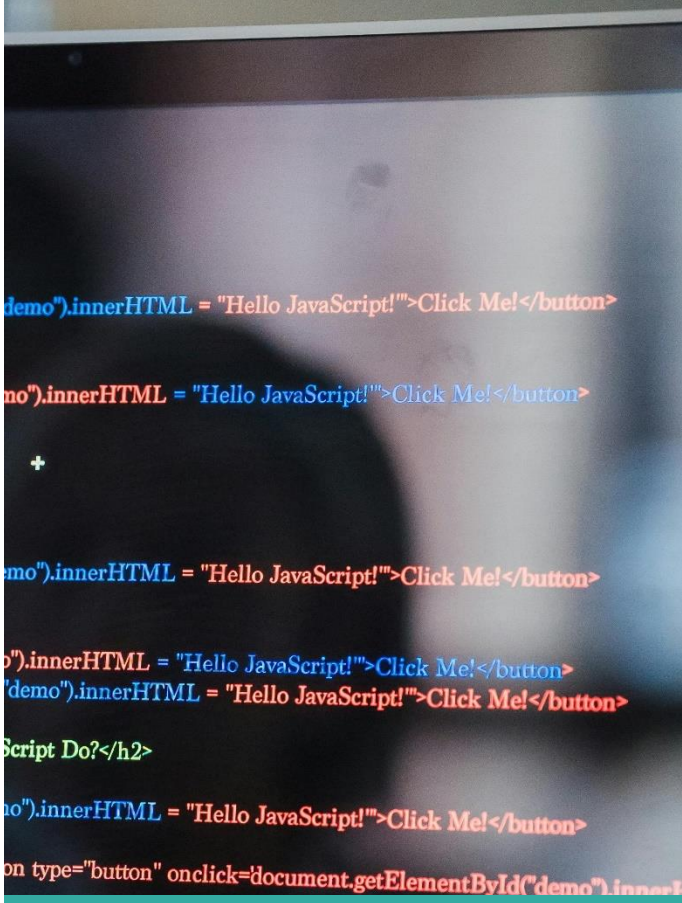
Lección 002



18 OCTUBRE

ROLOTECH

Creado por: Morales Campuzano Jose Rodrigo

Datatypes01.py

Código

```
# Cadenas de caracteres
cadena = "Hola Mundo!"
print("Esta es una cadena: ", cadena)

print("Primer caracter de la cadena: ", cadena[0])
print("Ultimo caracter de la cadena: ", cadena[-1])
```

Explicación

Este código:

1. Define una cadena llamada cadena.
2. Imprime la cadena completa.
3. Imprime el primer carácter de la cadena.
4. Imprime el último carácter de la cadena.

Salída

```
$ python datatypes01.py
Esta es una cadena:  Hola Mundo!
Primer caracter de la cadena:  H
Ultimo caracter de la cadena:  !
```

Datatypes02.py

Código

```
# Otras posiciones
cadena = "Hola Mundo!"
print("Esta es una cadena: ", cadena)

print("Posicion 0: ", cadena[0])
print("Posicion 1: ", cadena[1])
print("Posicion 2: ", cadena[2])
```

```
print("Posicion 3: ", cadena[3])
print("Posicion -1: ", cadena[-1])
print("Posicion -2: ", cadena[-2])
```

Explicación

Este código:

1. Define una cadena llamada cadena.
2. Imprime la cadena completa.
3. Imprime los caracteres en las posiciones 0, 1, 2, 3 de la cadena.
4. Imprime los caracteres en las posiciones -1 y -2 (últimos caracteres) de la cadena.

Salída

```
$ python datatypes02.py
Esta es una cadena:  Hola Mundo!
Posicion 0:  H
Posicion 1:  o
Posicion 2:  l
Posicion 3:  a
Posicion -1:  !
Posicion -2:  o
```

Datatypes03.py

Código

```
# Slicing
cadena = "Hola Mundo!"
print("Esta es una cadena: ", cadena)

print("Silice: ", cadena[0:4])
print("Silice: ", cadena[5:10])
print("Silice: ", cadena[5:])
print("Silice: ", cadena[:4])
print("Silice: ", cadena[-1:])
print("Silice: ", cadena[3:-2])
```

Explicación

Este código:

1. Define una cadena llamada cadena.
2. Imprime la cadena completa.
3. Realiza diferentes **slicing** (subcadenas) de la cadena en los siguientes rangos:
    - De la posición 0 a la 4 ("Hola").
    - De la posición 5 a la 10 ("Mundo").
    - Desde la posición 5 hasta el final ("Mundo!").
    - Desde el inicio hasta la posición 4 ("Hola").
    - Desde la última posición ("!").
    - Desde la posición 3 hasta dos antes del final ("a Mund").

Salída

```
$ python datatypes03.py
Esta es una cadena:  Hola Mundo!
Silice:  Hola
Silice:  Mundo
Silice:  Mundo!
Silice:  Hola
Silice:  !
Silice:  a Mund
```

Datatypes04.py

Código

```
# Invertir cadena
cadena = "Hola Mundo!"
print("Esta es una cadena: ", cadena)

print("Invertir: ", cadena[::-1])
```

Explicación

Este código:

1. Define una cadena llamada cadena.
2. Imprime la cadena completa.
3. Imprime la cadena invertida utilizando slicing con el paso [::-1].

Salída

```
$ python datatypes04.py
Esta es una cadena:  Hola Mundo!
Invertir:  !odnuM aloH
```

Datatypes05.py

Código

```
# Posicioes pares
cadena = "Hola Mundo!"
print("Esta es una cadena: ", cadena)

print("Posicion pares: ", cadena[::2])
```

Explicación

Este código:

1. Define una cadena llamada cadena.
2. Imprime la cadena completa.
3. Imprime los caracteres en posiciones pares de la cadena utilizando slicing con el paso ::2.

Salída

```
$ python datatypes05.py
Esta es una cadena:  Hola Mundo!
Posicion pares:  Hl ud!
```

Datatypes06.py

Código

```
# Que imprime?
cadena = "Hola Mundo!"
print("Esta es una cadena: ", cadena)

print("Puedes adivinar que esta imprimiento?: ", cadena[-1::-3])
```

Explicación

ste código:

1. Define una cadena llamada cadena.
2. Imprime la cadena completa.
3. Imprime caracteres de la cadena en orden invertido, saltando cada 3 posiciones (hacia atrás), comenzando desde el último carácter (!).

Resultado impreso: !nuoaH.

Salída

```
$ python datatypes06.py
Esta es una cadena:  Hola Mundo!
Puedes adivinar que esta imprimiento?:  !n o
```

Datatypes07.py

Código

```
# Otra forma de invertir una cadena
cadena = "Hola Mundo!"
print("Esta es una cadena: ", cadena)

cadenaInvertida = "".join(reversed(cadena))
print("Invertir 2da opcion: ", cadenaInvertida)
```

Explicación

Este código:

1. Define una cadena llamada cadena.
2. Imprime la cadena completa.
3. Usa la función reversed() para invertir la cadena y la convierte en una cadena unida con "".join().
4. Imprime la cadena invertida usando esta segunda opción.

Salída

```
$ python datatypes07.py
Esta es una cadena:  Hola Mundo!
Invertir 2da opcion:  !odnuM aloH
```

Datatypes08.py

Código

```
# Que esta pasando en este ejercicio?
cadena = "Hola Mundo!"
print("Esta es una cadena: ", cadena)

cadena[4] = '*'
print("Porque no puedo imprimir?: ", cadena)
```

Explicación

Este código intenta modificar un carácter en una cadena, lo cual no es posible porque las cadenas en Python son **inmutables** (no se pueden modificar directamente).

La línea cadena[4] = '*' produce un error porque intenta asignar un nuevo valor a una posición específica de la cadena.

Salída

```
$ python datatypes08.py
Esta es una cadena:  Hola Mundo!
Traceback (most recent call last):
```

```
  File
"E:\IPN\Superior\Semestres\Semestre_3\PCD\pcd3am1_2024\parcial1\leccion02\datatypes0
8.py", line 5, in <module>
    cadena[4] = '*'
    ~~~~~~^^^
TypeError: 'str' object does not support item assignment
```

Datatypes09.py

Código

```
# Para modificar una cadena podemos cambiarla completa
cadena = "Hola Mundo!"
print("Esta es una cadena: ", cadena)

cadena = "Hola*Mundo!"
print("Se modifico la cadena?: ", cadena)
```

Explicación

Este código:

1. Define una cadena llamada cadena.
2. Imprime la cadena completa.
3. Asigna una nueva cadena "Hola*Mundo!" a la variable cadena.
4. Imprime la cadena modificada con el asterisco en lugar del espacio.

Aquí la cadena se reemplaza por completo, lo cual es posible debido a que se está reasignando la variable, no modificando la cadena original.

Salída

```
$ python datatypes09.py
Esta es una cadena:  Hola Mundo!
Se modifico la cadena?:  Hola*Mundo!
```

Datatypes10.py

Código

```
# Para borrar un caracter de una cadena
cadena = "Hola Mundo!"
print("Esta es una cadena: ", cadena)

del cadena[4]
print("Se elimino el caracter?: ", cadena)
```

Explicación

Este código intenta eliminar un carácter de una cadena con del cadena[4], lo cual genera un error porque las cadenas en Python son **inmutables** y no se pueden modificar de esta manera.

El error ocurre porque no es posible eliminar un carácter directamente de una cadena. Para eliminar un carácter, se debe crear una nueva cadena sin ese carácter.

Salída

```
$ python datatypes10.py
Esta es una cadena:  Hola Mundo!
Traceback (most recent call last):
  File
"E:\IPN\Superior\Semestres\Semestre_3\PCD\pcd3am1_2024\parcial1\leccion02\datatypes1
0.py", line 5, in <module>
    del cadena[4]
        ~~~~~~^^^
TypeError: 'str' object doesn't support item deletion
```

Datatypes11.py

Código

```
# Para borrar un caracter de una cadena podemos hacer esto
cadena = "Hola Mundo!"
print("Esta es una cadena: ", cadena)
```

```
nuevaCadena = cadena[:4] + cadena[-1:]
print("Se eliminaron varios caracteres?: ", nuevaCadena)
```

Explicación

ste código:

1. Define una cadena llamada cadena.
2. Imprime la cadena completa.
3. Crea una nueva cadena, **nuevaCadena**, uniendo los primeros 4 caracteres (cadena[:4]) con el último carácter (cadena[-1:]), eliminando todo lo que está entre esos dos.
4. Imprime la nueva cadena, que solo contiene "Hola!", eliminando los caracteres intermedios.

Salída

```
$ python datatypes11.py
Esta es una cadena:  Hola Mundo!
Se eliminaron varios caracteres?:  Hola!
```

Datatypes12.py

Código

```
# Borrando una cadena completa
cadena = "Hola Mundo!"
print("Esta es una cadena: ", cadena)

del cadena
print("Cadena eliminada...")
print("Se elimino la cadena?: ", cadena)
```

Explicación

Este código:

1. Define una cadena llamada cadena.
2. Imprime la cadena completa.

3. Usa del cadena para eliminar la variable cadena por completo.
4. Intenta imprimir cadena nuevamente, lo cual generará un **error** porque la variable ya ha sido eliminada y no existe más.

Salída

```
$ python datatypes12.py
Esta es una cadena:  Hola Mundo!
Cadena eliminada...
Traceback (most recent call last):
  File
"E:\IPN\Superior\Semestres\Semestre_3\PCD\pcd3am1_2024\parcial1\leccion02\datatypes12.py", line 7, in <module>
    print("Se elimino la cadena?: ", cadena)
                                     ^^^^^^
NameError: name 'cadena' is not defined
```

Datatypes13.py

Código

```
# Formato de cadenas
cadena1 = "{} {} {}".format('Hola', 'Mundo', '!')
print("Esta es una cadena con formato: ", cadena1)

cadena2 = "{1} {2} {0}".format('Hola', 'Mundo', '!')
print("Esta es una cadena con formato desordenado: ", cadena2)

cadena3 = "{a} {b} {c}".format(a='Hola', b='Mundo', c='!')
print("Esta es una cadena con formato: ", cadena3)
```

Explicación

Este código utiliza el método format() para crear cadenas con formato:

1. **cadena1**: Inserta las palabras 'Hola', 'Mundo', y '!' en los espacios correspondientes de la cadena.
2. **cadena2**: Inserta las palabras en orden desordenado usando índices ({1}, {2}, {0}).
3. **cadena3**: Usa nombres de variables (a, b, c) para formatear la cadena con los valores 'Hola', 'Mundo', y '!'.

Salída

```
$ python datatypes13.py
Esta es una cadena con formato:  Hola Mundo !
Esta es una cadena con formato desordenado:  Mundo ! Hola
Esta es una cadena con formato:  Hola Mundo !
```

Datatypes14.py

Código

```
# Mas Formato de cadenas con numeros
cadena1 = "{0:b}".format(13)
print("Binario del 256: ", cadena1)

cadena2 = "{0:e}".format(1234.34534535359379)
print("Formato exponencial: ", cadena2)

cadena3 = "{a:.4f}".format(a=3.141592)
print("Flotante truncado a 4 digitos: ", cadena3)
```

Explicación

Este código usa el método format() para formatear números:

1. **cadena1**: Convierte el número 13 a formato **binario**.
2. **cadena2**: Convierte el número 1234.3453... a **formato exponencial**.
3. **cadena3**: Formatea el número 3.141592 truncándolo a **4 decimales** (3.1416).

Salída

```
$ python datatypes14.py
Binario del 256:  1101
Formato exponencial:  1.234345e+03
Flotante truncado a 4 digitos:  3.1416
```

Datatypes15.py

Código

```
# Alineando cadenas
String1 = "|{:<10}|{:^10}|{:>10}|".format('Izq',
                                'Cen',
                                'Der')
print("\nAlineacion de cadenas: ")
print(String1)

# Alineacion de espacios
String1 = "\n>>{0:^10}<< soy yo, y tengo >>{1:<4}<< años!".format("Mario",
                                    43)
print(String1)
```

Explicación

Este código utiliza el método format() para alinear cadenas:

1. **String1** (primera parte): Alinea tres palabras dentro de un espacio de 10 caracteres cada una:
    - '<10': Alinea a la izquierda.
    - '^10': Centra el texto.
    - '>10': Alinea a la derecha.

Resultado: |Izq | Cen | Der|.

2. **String1** (segunda parte): Alinea el texto y números en una cadena más compleja:
    - '^10': Centra la palabra "Mario" dentro de 10 espacios.
    - '<4': Alinea el número 43 a la izquierda dentro de 4 espacios.

Resultado: >> Mario << soy yo, y tengo >>43 << años!.

Salída

```
$ python datatypes15.py

Alineacion de cadenas:
|Izq      |   Cen    |      Der|
```

```
>>  Mario   << soy yo, y tengo >>43  << años
```

Datatypes16.py

Código

```
# ENteros y flotantes
entero = 5
flotante = 2/3

print(type(entero))
print(type(flotante))
print(entero)
print(flotante)
print(entero + flotante)
print(3+4)
print(3-4)
print(3*4)
print(3/4)
print(3//4)
print(3%4)
```

Explicación

Este código:

1. Define un **entero** y un **flotante**.
2. Imprime los tipos de datos de las variables entero y flotante.
3. Imprime los valores de las variables.
4. Realiza varias operaciones aritméticas:
    - Suma entero y flotante.
    - Suma, resta, multiplica y divide números.
    - Usa la división entera (//), que retorna la parte entera de la división.
    - Usa el operador módulo (%), que retorna el resto de la división.

Salída

```
$ python datatypes16.py
<class 'int'>
<class 'float'>
5
0.6666666666666666
5.666666666666667
7
-1
12
0.75
0
3
```

Datatypes17.py

Código

```
# Complejos

a = 1 + 5j
b = 2 + 3j

c = a + b
print("Suma:",c)

d = 1 + 5j
e = 2 - 3j

f = d - e
print("Resta:",f)

g = 1 + 5j
h = 2 + 3j

i = g / h
print("Division:",i)

j = 1 + 5j
k = 2 + 3j

l = j * k
print("Multiplicacion:",l)
```

Explicación

Este código trabaja con **números complejos**:

1. **Suma**:
    - a + b = (1 + 5j) + (2 + 3j) = 3 + 8j.
2. **Resta**:
    - d - e = (1 + 5j) - (2 - 3j) = -1 + 8j.
3. **División**:
    - (1 + 5j) / (2 + 3j) = 1.6153846153846154 + 1.076923076923077j.
4. **Multiplicación**:
    - (1 + 5j) * (2 + 3j) = -13 + 13j.

Los resultados muestran las operaciones básicas con números complejos.

Salída

```
$ python datatypes17.py
Suma: (3+8j)
Resta: (-1+8j)
Division: (1.307692307692308+0.5384615384615384j)
Multiplicacion: (-13+13j)
```

Datatypes18.py

Código

```
# Booleanos

a = True
b = False

print(type(a))

print(f'a={a}')

print(f'b={b}')
```

```
print(f'34 == 34 : {34==34}')

print(f'23 == 24: {23 == 24}')

print(f'34 != 34 : {34!=34}')

print(f'23 != 24: {23 != 24}')

x,y,z = 1,2,3

if x < y and y < z:
   print('x < y < z')
else:
   print('Error en el and')


if x > y or y < z:
   print('x > y  o  y < z')
else:
   print('Error en el or')

if not x > z:
   print('Negacion')
```

Explicación

Este código trabaja con **valores booleanos** y operadores lógicos:

1. **Tipos booleanos**: a es True y b es False. Se imprime el tipo de a (bool).
2. **Operadores de comparación**:
    - 34 == 34 es True.
    - 23 == 24 es False.
    - 34 != 34 es False.
    - 23 != 24 es True.
3. **Condicionales**:
    - El operador and en x < y and y < z evalúa True porque 1 < 2 < 3.
    - El operador or en x > y or y < z evalúa True porque y < z.

- El operador not niega la expresión x > z, lo que resulta en True ya que x no es mayor que z.

Salída

```
$ python datatypes18.py
<class 'bool'>
a=True
b=False
34 == 34 : True
23 == 24: False
34 != 34 : False
23 != 24: True
x < y < z
x > y  o  y < z
Negacion
```

Datatypes19.py

Código

```
# Listas

listaVacia = []

listaNumeros = [1,2,3,4,5,6]

listaLetras = ['a', 'b', 'c']

listaObjetos = [1, '*', 0, True, 3.141592, False, "Hola Mundo!", listaLetras]

print(f'Lista Vacia: {listaVacia}')
print(f'Tamaño Lista Vacia: {len(listaVacia)}')

print(f'Lista Numeros: {listaNumeros}')
print(f'Tamaño Lista Numeros: {len(listaNumeros)}')

print(f'Lista Letras: {listaLetras}')
print(f'Tamaño Lista Letras: {len(listaLetras)}')

print(f'Lista Objetos: {listaObjetos}')
print(f'Tamaño Lista Objetos: {len(listaObjetos)}')
```

Explicación

Este código define y trabaja con **listas**:

1. **Lista vacía** (listaVacia): No contiene elementos, su tamaño es 0.
2. **Lista de números** (listaNumeros): Contiene los números del 1 al 6, su tamaño es 6.
3. **Lista de letras** (listaLetras): Contiene los caracteres 'a', 'b', 'c', su tamaño es 3.
4. **Lista de objetos mixtos** (listaObjetos): Contiene distintos tipos de datos, incluyendo números, booleanos, cadenas, y otra lista (listaLetras), su tamaño es 8.

El código imprime las listas y sus tamaños usando len().

Salída

```
$ python datatypes19.py
Lista Vacia: []
Tamaño Lista Vacia: 0
Lista Numeros: [1, 2, 3, 4, 5, 6]
Tamaño Lista Numeros: 6
Lista Letras: ['a', 'b', 'c']
Tamaño Lista Letras: 3
Lista Objetos: [1, '*', 0, True, 3.141592, False, 'Hola Mundo!', ['a', 'b', 'c']]
Tamaño Lista Objetos: 8
```

Datatypes20.py

Código

```
# Listas Bidimensionales

matriz = [[1,2,3,4], [5,6,7,8], [9,10,11,12]]

print(f'Matriz completa: {matriz}')

print(f'Segundo Renglon: {matriz[1]}')

print(f'Tercer Renglon, segunda Columna: {matriz[2][1]}')
```

Explicación

Este código define y trabaja con una **matriz bidimensional** (lista de listas):

1. **Definición de la matriz** (matriz): Contiene tres listas, cada una representando un renglón.
2. Imprime la **matriz completa**, mostrando todos los renglones.
3. Imprime el **segundo renglón** (matriz[1]), que es [5, 6, 7, 8].
4. Imprime el **tercer renglón, segunda columna** (matriz[2][1]), que es 10.

Salída

```
$ python datatypes20.py
Matriz completa: [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
Segundo Renglon: [5, 6, 7, 8]
Tercer Renglon, segunda Columna: 10
```

Datatypes21.py

Código

```
# Listas acceso indexado negativo

lista = ['a', 'b', 'c', 'd', 'e']

print(f'Ultima posicion 1: {lista[4]}')
print(f'Ultima posicion 2: {lista[len(lista)-1]}')
print(f'Ultima posicion 3: {lista[-1]}')

print(f'Penultima posicion {lista[-2]}')
```

Explicación

Este código trabaja con **acceso indexado negativo** en una lista:

1. Define una lista llamada lista con los elementos ['a', 'b', 'c', 'd', 'e'].
2. Imprime la **última posición** de la lista de tres maneras:
    - Usando el índice positivo 4 (lista[4]), que devuelve 'e'.
    - Usando el tamaño de la lista len(lista) - 1, que también devuelve 'e'.
    - Usando el índice negativo -1 (lista[-1]), que devuelve 'e'.

3. Imprime la **penúltima posición** (lista[-2]), que devuelve 'd'.

Salída

```
python datatypes21.py
Ultima posicion 1: e
Ultima posicion 2: e
Ultima posicion 3: e
Penultima posicion d
```

Datatypes22.py

Código

```
# Cadena a Lista

cadena = 'Hola Mundo! Aqui estoy ...'
lista = cadena.split()

print(f'Separada por espacios en una lista: {lista}')

cadena2 = "1&2&3&4"
print(f'Otro separador: {cadena2.split('&')}')
```

Explicación

Este código convierte cadenas a listas usando el método split():

1. **Cadena inicial**: Define una cadena cadena con el valor 'Hola Mundo! Aqui estoy ...'.
    - Usa split() sin argumentos para separar la cadena en una lista de palabras, separadas por espacios.
    - Imprime la lista resultante: ['Hola', 'Mundo!', 'Aqui', 'estoy', '...'].
2. **Otra cadena**: Define cadena2 con el valor "1&2&3&4".
    - Usa split('&') para separar la cadena en una lista usando & como separador.
    - Imprime la lista resultante: ['1', '2', '3', '4'].

4o mini

Salída

```
$ python datatypes22.py
Separada por espacios en una lista: ['Hola', 'Mundo!', 'Aqui', 'estoy', '...']
Otro separador: ['1', '2', '3', '4']
```

Datatypes23.py

Código

```
# Agregando a Listas

# Primero es una lista vacia
lista = []
print(f'Lista vacia: {lista}')

lista.append('IPN')
lista.append('UNAM')
lista.append('TEC')
lista.append('IBERO')

print(f'Universidades: {lista}')

for i in range(2,5):
    lista.append(i)

print(f'Lista con universidades y numeros: {lista}')

lista.insert(2, 'ANAHUAC')
print(f'Insertando un objeto: {lista}')
```

Explicación

Este código trabaja con listas y muestra cómo agregar elementos a ellas:

1. **Lista vacía**: Se define una lista lista vacía y se imprime.
2. **Agregar elementos**: Se utilizan append() para agregar varias universidades a la lista:
    - Se agregan 'IPN', 'UNAM', 'TEC', y 'IBERO'.
    - Se imprime la lista con universidades: ['IPN', 'UNAM', 'TEC', 'IBERO'].
3. **Agregar números**: Se utiliza un bucle for para agregar los números del 2 al 4 (inclusive) a la lista.

- La lista ahora incluye tanto universidades como números: ['IPN', 'UNAM', 'TEC', 'IBERO', 2, 3, 4].

4. **Insertar un elemento**: Se usa insert(2, 'ANAHUAC') para insertar 'ANAHUAC' en la posición 2 de la lista.
    - Se imprime la lista final: ['IPN', 'UNAM', 'ANAHUAC', 'TEC', 'IBERO', 2, 3, 4].

Salída

```
$ python datatypes23.py
Lista vacia: []
Universidades: ['IPN', 'UNAM', 'TEC', 'IBERO']
Lista con universidades y numeros: ['IPN', 'UNAM', 'TEC', 'IBERO', 2, 3, 4]
Insertando un objeto: ['IPN', 'UNAM', 'ANAHUAC', 'TEC', 'IBERO', 2, 3, 4]
```

Datatypes24.py

Código

```
# Invirtiendo una lista

lista = ['IPN', 'UNAM', 'ANAHUAC', 'TEC', 'IBERO', 2, 3, 4]

print(f'Lista original: {lista}')

lista.reverse()

print(f'Lista invertida: {lista}')

print(f'Lista original nuevamente: {lista[::-1]}')

print(f'Volvemos a invertirla: {list(reversed(lista))}')
```

Explicación

Este código invierte una lista y muestra diferentes formas de hacerlo:

1. **Lista original**: Se define una lista lista con varias universidades y números.
2. Se imprime la lista original: ['IPN', 'UNAM', 'ANAHUAC', 'TEC', 'IBERO', 2, 3, 4].
3. **Invertir la lista**:

4. Se utiliza lista.reverse() para invertir la lista en su lugar.
5. Se imprime la lista invertida: [4, 3, 2, 'IBERO', 'TEC', 'ANAHUAC', 'UNAM', 'IPN'].
6. **Mostrar la lista original nuevamente**:
7. Se imprime la lista original usando el **slicing** lista[::-1], que devuelve la lista invertida sin modificar la original.
8. **Invertir de nuevo**:
9. Se utiliza list(reversed(lista)) para crear una nueva lista invertida sin cambiar la lista original, que también se imprime.
10. Este último resultado muestra que la lista ha vuelto a su orden original: ['IPN', 'UNAM', 'ANAHUAC', 'TEC', 'IBERO', 2, 3, 4].

Salída

```
$ python datatypes24.py
Lista original: ['IPN', 'UNAM', 'ANAHUAC', 'TEC', 'IBERO', 2, 3, 4]
Lista invertida: [4, 3, 2, 'IBERO', 'TEC', 'ANAHUAC', 'UNAM', 'IPN']
Lista original nuevamente: ['IPN', 'UNAM', 'ANAHUAC', 'TEC', 'IBERO', 2, 3, 4]
Volvemos a invertirla: ['IPN', 'UNAM', 'ANAHUAC', 'TEC', 'IBERO', 2, 3, 4]
```

Datatypes25.py

Código

```
# Eliminando elementos de un alista

lista = [1, 2, 3, 4, 2, 2, 5, 6, 2, 7]

print(f'Lista original: {lista}')

lista.remove(2)
lista.remove(6)

print(f'Se eleimino un elemento: {lista}')

x = lista.pop()

print(f'Se extrajo el ultimo elemento: {x} de la lista: {lista}')
```

Explicación

Este código muestra cómo eliminar elementos de una lista:

1. **Lista original**: Se define una lista lista con varios números, que incluye múltiples ocurrencias del número 2.
    - Se imprime la lista original: [1, 2, 3, 4, 2, 2, 5, 6, 2, 7].
2. **Eliminar elementos**:
    - Se utiliza lista.remove(2) para eliminar la primera ocurrencia del número 2.
    - Se utiliza lista.remove(6) para eliminar el número 6.
    - Se imprime la lista después de las eliminaciones: [1, 3, 4, 2, 2, 5, 2, 7].
3. **Extraer el último elemento**:
    - Se utiliza x = lista.pop() para extraer y eliminar el último elemento de la lista (7).
    - Se imprime el valor extraído (7) y la lista resultante: [1, 3, 4, 2, 2, 5, 2].

Salída

```
$ python datatypes25.py
Lista original: [1, 2, 3, 4, 2, 2, 5, 6, 2, 7]
Se eleimino un elemento: [1, 3, 4, 2, 2, 5, 2, 7]
Se extrajo el ultimo elemento: 7 de la lista: [1, 3, 4, 2, 2, 5, 2]
```

Datatypes26.py

Código

```
# Slicing

lista = [i for i in range(100,1100, 100)]

print(f'Lista original: {lista}')
```

```
print(f'Slice de 0 a 9: {lista[0:len(lista)]}')

print(f'Slice de 0 a 2: {lista[0:3]}')

print(f'Slice de Inicio a 2: {lista[:3]}')

print(f'Slice de 3 a 9: {lista[3:10]}')

print(f'Slice de 3 al final: {lista[3:]}')

print(f'Slice de 4 a 6: {lista[4:7]}')

print(f'Slice de Inicio a Fin: {lista[:]}')

print(f'Slice posiciones pares: {lista[::2]}')

print(f'Slice posiciones impares: {lista[1::2]}')

print(f'Slice invertido impares: {lista[-1::-2]}')

print(f'Slice invertido pares: {lista[-2::-2]}')
```

Explicación

Este código utiliza **slicing** para acceder a diferentes partes de una lista:

1. **Lista original**: Se crea una lista lista que contiene los números del 100 al 1000 en incrementos de 100: [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000].
    - Se imprime la lista original.
2. **Diferentes slices**:
    - **De 0 a 9**: lista[0:len(lista)] devuelve la lista completa.
    - **De 0 a 2**: lista[0:3] devuelve los primeros tres elementos: [100, 200, 300].
    - **De inicio a 2**: lista[:3] también devuelve los primeros tres elementos: [100, 200, 300].
    - **De 3 a 9**: lista[3:10] devuelve los elementos desde el índice 3 al final: [400, 500, 600, 700, 800, 900, 1000].
    - **De 3 al final**: lista[3:] devuelve los elementos desde el índice 3 en adelante: [400, 500, 600, 700, 800, 900, 1000].

- **De 4 a 6**: lista[4:7] devuelve los elementos desde el índice 4 hasta el índice 6: [500, 600, 700].
- **De inicio a fin**: lista[:] devuelve la lista completa.
- **Posiciones pares**: lista[::2] devuelve los elementos en las posiciones pares: [100, 300, 500, 700, 900].
- **Posiciones impares**: lista[1::2] devuelve los elementos en las posiciones impares: [200, 400, 600, 800, 1000].
- **Invertido impares**: lista[-1::-2] devuelve los elementos en las posiciones impares, pero en orden invertido: [1000, 800, 600, 400, 200].
- **Invertido pares**: lista[-2::-2] devuelve los elementos en las posiciones pares en orden invertido: [900, 700, 500, 300, 100].

4o mini

Salída

```
Lista original: [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
Slice de 0 a 9: [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
Slice de 0 a 2: [100, 200, 300]
Slice de Inicio a 2: [100, 200, 300]
Slice de 3 a 9: [400, 500, 600, 700, 800, 900, 1000]
Slice de 3 al final: [400, 500, 600, 700, 800, 900, 1000]
Slice de 4 a 6: [500, 600, 700]
Slice de Inicio a Fin: [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]
Slice posiciones pares: [100, 300, 500, 700, 900]
Slice posiciones impares: [200, 400, 600, 800, 1000]
Slice invertido impares: [1000, 800, 600, 400, 200]
Slice invertido pares: [900, 700, 500, 300, 100]
```