Qué función colocaría en la línea 1 para que el resultado que imprime la línea 2 sea <class 'str'>

1. x = 5

2. print(type(x))

Python tiene los siguientes tipos de datos integrados de forma predeterminada, en estas categorías:

* Tipo texto => str
* Tipos numéricos => int, float, complex
* Tipos secuencia => list, tuple, **range**
* Tipo asignación => dict
* Tipos conjunto => set
* Tipo booleano => bool

Puede obtener el tipo de datos de cualquier objeto utilizando la función type():

print(type(x))

Si desea especificar el tipo de datos, puede utilizar las siguientes funciones de construcción:

|  |  |
| --- | --- |
| **Example** | **Data Type** |
| x = str("Hello World") | str |
| x = int(20) | int |
| x = float(20.5) | float |
| x = complex(1j) | complex |
| x = list(("apple", "banana", "cherry")) | list |
| x = tuple(("apple", "banana", "cherry")) | Tuple |
| x = range(6) | Range |
| x = dict(name="John", age=36) | Dict |
| x = set(("apple", "banana", "cherry")) | Set |
| x = frozenset(("apple", "banana", "cherry")) | Frozenset |
| x = bool(5) | Bool |
| x = bytes(5) | Bytes |
| x = bytearray(5) | Bytearray |
| x = memoryview(bytes(5)) | Memoryview |

**Respuesta:**

x = '5' ó x = str(5)

[**https://www.w3schools.com/python/python\_datatypes.asp**](https://www.w3schools.com/python/python_datatypes.asp)

Mencione los tipos numéricos que posee Python y vistos en clase

Python tiene los siguientes tipos de datos integrados de forma predeterminada, en estas categorías:

* Tipo texto => str
* Tipos numéricos => int, float, complex
* Tipos secuencia => list, tuple, **range**
* Tipo asignación => dict
* Tipos conjunto => set
* Tipo booleano => bool

**Respuesta:**

int, float, complex

[**https://www.w3schools.com/python/python\_datatypes.asp**](https://www.w3schools.com/python/python_datatypes.asp)

Considerando los tipos de secuencia y si decimos que es una colección ordenada e inmutable nos referimos a:

* Tipos secuencia => list, tuple, range
* Tipo asignación => dict
* Tipo conjunto => set

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | ¿Ordenado? | ¿Mutable? | ¿Duplicados? |
| **list** | [] | Sí | Sí | Sí |
| **tuple** | () | Sí | **No** | Sí |
| **set** | **{}** | **No** | Sí | **No** |
| **dict** | **{}** | Sí | Sí | **No** |

**Respuesta:**

Tupla

[**https://www.w3schools.com/python/python\_tuples.asp**](https://www.w3schools.com/python/python_tuples.asp)

Qué se observa en la consola si ejecutamos el siguiente código:

numeros = (1, 2, 3)

resultado = numeros \* 2

print(resultado)

|  |  |  |
| --- | --- | --- |
|  | join | multiply |
| **dict** |  |  |
| **list** | list3 = list1 + list2 |  |
| **set** | set3 = set1.union(set2) |  |
| **tuple** | tuple3 = tuple1 + tuple2 | mytuple = fruits \* 2 |

Si desea multiplicar el contenido de una tupla un número determinado de veces, puede utilizar el operador \*.

**Ejemplo.** Multiplica la tupla de frutas por 2:

fruits = ("apple", "banana", "cherry")  
mytuple = fruits \* 2  
print(mytuple)

# Salida => ('apple', 'banana', 'cherry', 'apple', 'banana', 'cherry')

**Respuesta:**

(1, 2, 3, 1, 2, 3)

[**https://www.w3schools.com/python/python\_tuples\_join.asp**](https://www.w3schools.com/python/python_tuples_join.asp)

Considerando los tipos de secuencia y si decimos que es una colección desordenada, inmutable y no indexada nos referimos a:

(Entendemos por inmutable a la no posibilidad de realizar cambios pero sí a eliminar y agregar datos)

**Conjunto (Set)**

Los conjuntos se utilizan para almacenar varios elementos en una sola variable. Un conjunto es una colección desordenada, inalterable\* y no indexada. \* Nota: Los elementos establecidos no se pueden modificar, pero puede eliminar elementos y agregar elementos nuevos. Los conjuntos se escriben con corchetes.

Agregar Elementos

Add "orange" by using the add() method:

thisset = {"apple", "banana", "cherry"}  
 thisset.add("orange")

Eliminar Elementos

Remove "banana" by using the remove() method:

thisset = {"apple", "banana", "cherry"}  
 thisset.remove("banana")

**Colecciones de Python (matrices/arrays)**

Hay cuatro tipos de datos de recopilación en el lenguaje de programación Python:

* La **lista** es una colección ordenada y modificable. Permite miembros duplicados.
* La **tupla** es una colección ordenada e inmutable. Permite miembros duplicados.
* El **conjunto** es una colección desordenada, inmutable**\*** y no indexada. No hay miembros duplicados.
* El diccionario es una colección ordenada**\*\*** y modificable. No hay miembros duplicados.

**\***Los elementos establecidos no se pueden cambiar, pero puede eliminar y/o agregar elementos cuando lo desee.

**\*\***A partir de la versión 3.7 de Python, se ordenan los diccionarios. En Python 3.6 y versiones anteriores, los diccionarios están desordenados.

Al elegir un tipo de colección, es útil comprender las propiedades de ese tipo. Elegir el tipo correcto para un conjunto de datos en particular podría significar la retención del significado y podría significar un aumento en la eficiencia o la seguridad.

**Respuesta:**

El conjunto (set)

[**https://www.w3schools.com/python/python\_sets.asp**](https://www.w3schools.com/python/python_sets.asp)

Indique que colocaría en las líneas punteadas para observar 2007:

child1 = {

"name" : "Emil",

"year" : 2004

}

child2 = {

"name" : "Tobias",

"year" : 2007

}

child3 = {

"name" : "Linus",

"year" : 2011

}

myfamily = {

"child1" : child1,

"child2" : child2,

"child3" : child3

}

print(myfamily.................)

## Access Nested Dictionary Items

You can access individual items in a nested dictionary by specifying key in multiple square brackets.

D = {'emp1': {'name': 'Bob', 'job': 'Mgr'},

'emp2': {'name': 'Kim', 'job': 'Dev'},

'emp3': {'name': 'Sam', 'job': 'Dev'}}

print(D['emp1']['name'])

# Prints Bob

print(D['emp2']['job'])

# Prints Dev

If you refer to a key that is not in the nested dictionary, an exception is raised.

print(D['emp1']['salary'])

# Triggers KeyError: 'salary'

To avoid such exception, you can use the special dictionary [get()](https://www.learnbyexample.org/python-dictionary-get-method/) method. This method returns the value for key if key is in the dictionary, else None, so that this method never raises a KeyError.

# key present

print(D['emp1'].get('name'))

# Prints Bob

# key absent

print(D['emp1'].get('salary'))

# PrintsNone

La diferencia en la "apariencia" entre un json y un dictionary:

import json

# Un JSON:

jayson = '{"name": "John", "age": 30, "city": "New York"}'

# Procesar jayson:

diccionario = json.loads(jayson)

# El resultado es un diccionario:

print(diccionario)

# Salida: {'name': 'John', 'age': 30, 'city': 'New York'}

**Respuesta:**

print(myfamily["child2"]["year"])

print(myfamily["child2"].get("year"))

[**https://www.w3schools.com/python/gloss\_python\_nested\_dictionaries.asp**](https://www.w3schools.com/python/gloss_python_nested_dictionaries.asp)

Complete que pondría en 1,2 y 3 para poder ver números desde el 0 hasta el 5

for ....1....... in range(......2......):

print(.....3.......)

**Esto es remarcable sobre range():**

¿Cómo iterar una lista buscando borrar algunos ítems sin que de error de índice desbordado?

for i in range(len(materias) - 1, -1, -1):

if int(notas[i]) >= 7:

materias.pop(i)

notas.pop(i)

Se comprende mejor viendo la sintaxis:

## Syntax

range(start, stop, step)

## Parameter Values

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| start | Optional. An integer number specifying at which position to start. **Default is 0** |
| stop | **Required.** An integer number specifying at which position to stop (**not included**). |
| step | Optional. An integer number specifying the incrementation. **Default is 1** |

En el ejemplo (Listas2 de la práctica), el len de materias es cinco:

for i in range((5) - 1, -1, -1):

print(i, end=" ")

# Salida: 4 3 2 1 0

Es decir, ésta es la forma de trabajar “in situ” con la lista (eliminando de atrás hacia adelante). La consigna pedía eliminar de la lista (la opción más usual es trabajar con una copia, pero ésta solución es más elegante y responde con exactitud a la consigna: “eliminar de la lista”).

**Funciones integradas de Python**

Python tiene un conjunto de funciones integradas. range() devuelve una secuencia de números, a partir de 0 y se incrementa en 1 (por defecto), y se detiene antes de un número específico.

Sintaxis: rango (inicio, parada, paso)

**Respuesta:**

for x in range(0, 6, 1):

print(x)

[**https://www.w3schools.com/python/ref\_func\_range.asp**](https://www.w3schools.com/python/ref_func_range.asp)

Complete que pondría en 1, 2, 3, 4, 5 y 6 para poder ver el doble del valor ingresado en 6

def myfunc(1):

return lambda 2 : 3 \* 4

mydoubler = myfunc(5)

print(mydoubler(6))

**Convertir funciones a funciones anónimas lamba**

Función

def mult(x, y):

return x \* y

Lambda

multiplica = lambda x, y: x \* y

print(multiplica(2, 4))

Ejemplo

def doble(x):

return x \* 2

lista1 = [1, 2, 3, 4, 5, 6]

lista2 = list(map(doble, lista1))

print(lista2)

# Output: [2, 4, 6, 8, 10, 12]

Lambda

lista1 = [1, 2, 3, 4, 5, 6]

lista2 = list(map(lambda x: x \* 2, lista1))

print(lista2)

# Output: [2, 4, 6, 8, 10, 12]

Ejemplo filtrado con lambda

lista1 = [18, -3, 5, 0, -1, 12]

lista2 = list(filter(lambda x: x > 0, lista1))

print(lista2)

# Output: [18, 5, 12]

**Ahora, “el ejemplo”**

En lugar de crear múltiples funciones, puedes crear una sola función **multiplicar\_por()** y llamarla con diferentes argumentos para crear una función que duplique o triplique.

def multiplicar\_por (**n**):

return lambda x: x \* **n**

duplicar = multiplicar\_por(2)

triplicar = multiplicar\_por(3)

diez\_veces = multiplicar\_por(10)

La función lambda toma el valor de **n** de la función **multiplicar\_por(n)** así que en **duplicar** el valor de **n** es **2**, en **triplicar** **n** vale **3** y en **diez\_veces** vale **10**. Y al llamar a estas funciones con un argumento podemos retornar el número multiplicado:

duplicar(6)

> 12

triplicar(5)

> 15

diez\_veces(12)

> 120

**¿Por qué utilizar las funciones de Lambda?**

El poder de lambda se muestra mejor cuando los usa como una función anónima dentro de otra función[[1]](#footnote-1).

Digamos que tiene una definición de función que toma un argumento, y ese argumento se multiplicará con un número desconocido:

def myfunc(n):  
 return lambda a : a \* n

Use esa definición de función para hacer una función que siempre duplique el número que envía:

Una función lambda es una pequeña función anónima.

Una función lambda puede tomar cualquier cantidad de argumentos, pero solo puede tener una expresión.

**Respuesta:**

def myfunc(n):  
 return lambda a : a \* n

mydoubler = myfunc(2)

print(mydoubler(11))

[**https://www.w3schools.com/python/python\_lambda.asp**](https://www.w3schools.com/python/python_lambda.asp)

Complete la clase Person para que posea la función constructora de acuerdo al código que se observa más abajo

class Person:

................................................

...............................................

p1 = Person("John", 36)

print(p1.name)

print(p1.age)

**Respuesta:**

class Person:  
 def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
 p1 = Person("John", 36)

print(p1)

[**https://www.w3schools.com/python/python\_classes.asp**](https://www.w3schools.com/python/python_classes.asp)

Complete la línea de puntos para que se vea 30:

import json

x = '{ "name":"John", "age":30, "city":"New York"}'

y = json...........................(x)

print(y["age"])

**Respuesta:**

import json

x = '{ "name":"John", "age":30, "city":"New York"}'

y = json.loads(x)

print(y["age"])

[**https://www.w3schools.com/python/python\_classes.asp**](https://www.w3schools.com/python/python_classes.asp)

1. Los ejemplos map y filter son un excelente ejemplo de eso. [↑](#footnote-ref-1)