

El alumnado crea archivo en formato PDF donde se evidencia respuestas a cada ejercicio de todas las secciones RESUMEN DE LA SECCION, el nombre del archivo deber tener como nombre U1R1_Ejer2_[nombre estudiante].PDF

Puntos Claves

- Si deseas importar un módulo como un todo, puedes hacerlo usando la sentencia `import nombre_del_módulo`. Puedes importar más de un módulo a la vez utilizando una lista separada por comas. Por ejemplo:

```
import mod1
import mod2, mod3, mod4
```

Aunque la última forma no se recomienda por razones estilísticas, y es mejor y más bonito expresar la misma intención de una forma más detallada y explícita, como por ejemplo:

```
import mod2
import mod3
import mod4
```

- Si un módulo se importa de la manera anterior y deseas acceder a cualquiera de sus entidades, debes anteponer el nombre de la entidad empleando la **notación con punto**. Por ejemplo:

```
import my_module
result = my_module.my_function(my_module.my_data)
```

El fragmento de código utiliza dos entidades que provienen del módulo `my_module`: una función llamada `my_function()` y una variable con el nombre `my_data`. Ambos nombres **deben tener el prefijo** `my_module.`. Ninguno de los nombres de entidad importados entra en conflicto con los nombres idénticos existentes en el namespace de tu código.

- Se te permite no solo importar un módulo como un todo, sino también importar solo entidades individuales de él. En este caso, las entidades importadas **no deben especificar** el prefijo cuando son empleadas. Por ejemplo:

Ejercicio 1

Quieres invocar la función `make_money()` contenida en el módulo llamado `mint`. Tu código comienza con la siguiente linea:

```
import mint
```

¿Cuál es la forma adecuada de invocar a la función?

Ejercicio 2

Quieres invocar la función `make_money()` contenida en el módulo llamado `mint`.

```
from mint import make_money
```

¿Cuál es la forma adecuada de invocar a la función?

Ejercicio 3

Quieres invocar la función `make_money()` contenida en el módulo llamado `mint`.

```
make_money()
```

Plataforma E

Universidad Tecnológica de Panamá

Plataforma E

elearn...

AXEL ALEJANDRO ALMAGUER RODRIGUEZ

Área personal

Perfil

Calificaciones

Mensajes

Preferencias

GUAN

Cerrar sesión

Puntos Clave

- Una función llamada `dir()` puede mostrar una lista de las entidades contenidas dentro de un módulo importado. Por ejemplo:

```
import os
dir(os)
```

Imprime una lista de todo el contenido del módulo `os` el cual, puedes usar en tu código.

- El módulo `math` contiene más de 50 funciones y constantes que realizan operaciones matemáticas (como `sine()`, `pow()`, `factorial()`) o aportando valores importantes (como `π` y la constante de Euler `e`).
- El módulo `random` agrupa más de 60 entidades diseñadas para ayudarte a usar números pseudoaleatorios. No olvides el prefijo "pseudo", ya que no existe un número aleatorio real cuando se trata de generarlos utilizando los algoritmos de la computadora.
- El módulo `platform` contiene alrededor de 70 funciones que te permiten sumergirte en las capas subyacentes del sistema operativo y el hardware. Usarlos te permite aprender más sobre el entorno en el que se ejecuta tu código.
- El **Índice de Módulos de Python** (<https://docs.python.org/3/py-modindex.html>) es un directorio de módulos impulsado por la comunidad disponible en el universo de Python. Si deseas encontrar un módulo que se adapte a tus necesidades, comienza tu búsqueda allí.

Ejercicio 1

¿Cuál es el valor esperado de la variable `result` después de que se ejecuta el siguiente código?

```
import math
result = math.e == math.exp(1)
```

Ejercicio 2

(Completa el enunciado) Establecer la semilla del generador de números pseudoaleatorios.

Ejercicio 3

¿Cuál de las funciones del módulo `platform` utilizarías para obtener la información de procesamiento?

Plataforma E

Universidad Tecnológica de Panamá

Plataforma E

elearn...

AXEL ALEJANDRO ALMAGUER RODRIGUEZ

Área personal

Perfil

Calificaciones

Mensajes

Preferencias

GUAN

Cerrar sesión

INSTITUTE

MODULE (67%) SECTION (100%)

Puntos Clave

- Mientras que un **módulo** está diseñado para acopiar algunas entidades relacionadas como funciones, variables o constantes, un **paquete** es un contenedor que permite el acoplamiento de varios módulos relacionados bajo un mismo nombre. Dicho contenedor se puede distribuir tal cual (como un lote de archivos implementados en un subárbol de directorio) o se puede empaquetar dentro de un archivo zip.
- Durante la primera importación del módulo, Python traduce su código fuente a un formato **semi-compilado** almacenado dentro de los archivos **pyc** y los implementa en el directorio **__pycache__** ubicado en el directorio de inicio del módulo.
- Si deseas decirle al usuario del módulo que una entidad en particular debe tratarse como **privada** (es decir, no debe usarse explícitamente fuera del módulo), puedes marcar su nombre con el prefijo `_` o `__`. No olvides que esto es solo una recomendación, no una orden.
- Los nombres **shabang**, **shebang**, **hasbang**, **poundbang** y **hashpling** describen el dígrafo escrito como `#!`, se utiliza para instruir a los sistemas operativos similares a Unix sobre cómo se debe iniciar el archivo fuente de Python. Esta convención no tiene efecto en MS Windows.
- Si deseas convencer a Python de que debe tomar en cuenta el directorio de un paquete no estándar, su nombre debe insertarse/agregarse en/a la lista de directorios de importación almacenada en la variable `path` contenida en el módulo `sys`.
- Un archivo de Python llamado `__init__.py` se ejecuta implícitamente cuando un paquete que lo contiene está sujeto a importación y se utiliza para inicializar un paquete y/o sus subpaquetes (si los hay). El archivo puede estar vacío, pero no debe faltar.

Ejercicio 1
Deseas evitar que el usuario de tu módulo ejecute tu código como un script ordinario. ¿Cómo lograrías tal efecto?

Revisar

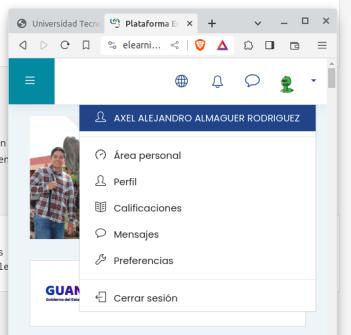
```
import sys
if __name__ == "__main__":
    print("No hagas eso!")
    sys.exit()
```

Ejercicio 2
Algunos paquetes adicionales y necesarios se almacenan asegurándose de que Python recorra el directorio para encontrarlos.

Revisar

```
import sys
# ¡Toma en cuenta las diagonales invertidas!
sys.path.append("D:\\Python\\Project\\Module")
```

Ejercicio 3



PYTHON INSTITUTE

MODULE (95%) SECTION (95%)

Puntos Claves

- Un **repositorio** (o **repo**) diseñado para recopilar y compartir código Python gratuito lleva por nombre **Python Package Index (PyPI)** aunque también es probable que te encuentres con el nombre de **The Cheese Shop** (La Tienda de Queso). Su sitio web está disponible en <https://pypi.org/>.
- Para hacer uso de The Cheese Shop, se ha creado una herramienta especializada y su nombre es **pip** (**pip Instala paquetes** mientras que **pip** significa ... ok, no importa). Como es posible que pip no se implemente como parte de la instalación estándar de Python, es posible que debas instalarlo manualmente. Pip es una herramienta de consola.
- Para verificar la versión de pip, se deben emitir los siguientes comandos:

```
pip --version
o
pip3 --version
```

Comprueba tu mismo cuál de estos funciona en el entorno de tu sistema operativo.

- La lista de las actividades principales de **pip** tiene el siguiente aspecto:
 - `pip help operación_o_comando` muestra una breve descripción de `pip`.
 - `pip list` muestra una lista de los paquetes instalados actualmente.
 - `pip show nombre_de_paquete` muestra información que incluyen las dependencias del paquete.
 - `pip search cadena` busca en los directorios de PyPI para encontrar paquetes cuyos nombres contengan `cadena`.

Ejercicio 1
¿De dónde proviene el nombre *The Cheese Shop*?

Revisar

Es una referencia a un viejo sketch de *Monty Python* que lleva el mismo nombre.

Ejercicio 2
¿Por qué deberías asegurarte de cuál `pip` o `pip3` es el correcto?

Revisar

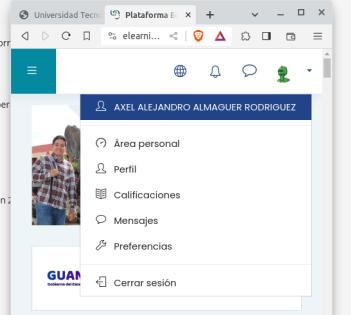
Cuando Python 2 y Python 3 coexisten en el sistema operativo, es importante saber cuál de los dos es el correcto para ejecutar paquetes de Python 2.

Ejercicio 3
¿Cómo puedes determinar si tu `pip` funciona con Python 2?

Revisar

`pip --version` te lo dirá.

Ejercicio 4



PYTHON INSTITUTE

MODULE (74%) SECTION (100%)

Puntos Clave

- Las computadoras almacenan caracteres como números. Hay más de una forma posible de codificar caracteres, pero solo algunas de ellas ganaron popularidad en todo el mundo y se usan comúnmente en TI: estas son **ASCII** (se emplea principalmente para codificar el alfabeto latino y algunos de sus derivados) y **UNICODE** (capaz de codificar prácticamente todos los alfabetos que utilizan los seres humanos).
- Un número correspondiente a un carácter en particular se llama **punto de código**.
- UNICODE utiliza diferentes formas de codificación cuando se trata de almacenar los caracteres usando archivos o memoria de computadora: dos de ellas son **UCS-4** y **UTF-8** (esta última es la más común ya que desperdicia menos espacio de memoria).

Ejercicio 1
¿Qué es BOM?

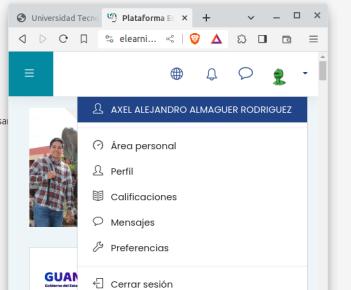
Revisar

BOM (Byte Order Mark). Una Marca de Orden de Bytes es una combinación especial de bits que anuncia la codificación utilizada por el contenido de un archivo (por ejemplo, UCS-4 o UTF-8).

Ejercicio 2
¿Está Python 3 internacionalizado?

Revisar

Si, está completamente internacionalizado: podemos usar los mismos caracteres y enviarlos a la salida.



PYTHON INSTITUTE

MODULE (24%) SECTION (100%)

Puntos Claves

1. Las cadenas de Python son **secuencias inmutables** y se pueden indexar, dividir en rebanadas e iterar como cualquier otra secuencia, además de estar sujetas a los operadores `in` y `not in`. Existen dos tipos de cadenas en Python:

- Cadenas de **una línea**, las cuales no pueden cruzar los límites de una línea, las denotamos usando apóstrofes (`'cadena'`) o comillas (`"cadena"`).
- Cadenas **multilinea**, que ocupan más de una línea de código fuente, delimitadas por apóstrofes triples:

```
'''  
cadena  
'''  
  
o  
  
'''  
cadena  
'''
```

2. La longitud de una cadena está determinada por la función `len()`. El carácter de escape (`\`) no es contado. Por ejemplo:

```
print(len("\n\n"))
```

Ejercicio 1
¿Cuál es la longitud de la siguiente cadena asumiendo que cada carácter tiene un espacio?

```
'''  
'''
```

Ejercicio 2
¿Cuál es el resultado esperado del siguiente código?

```
s = 'yesteryears'  
the_list = list(s)  
print(the_list[3:6])
```

Ejercicio 3
¿Cuál es el resultado esperado del siguiente código?

```
print(len("\n\n"))
```

PYTHON INSTITUTE

MODULE (45%) SECTION (94%)

Puntos Clave

1. Algunos de los métodos que ofrecen las cadenas son:

- `capitalize()`: cambia todas las letras de la cadena a mayúsculas.
- `center()`: centra la cadena dentro de una longitud conocida.
- `count()`: cuenta las ocurrencias de un carácter dado.
- `join()`: une todos los elementos de una tupla/lista en una cadena.
- `lower()`: convierte todas las letras de la cadena en minúsculas.
- `lstrip()`: elimina los caracteres en blanco al principio de la cadena.
- `replace()`: reemplaza una subcadena dada con otra.
- `rfind()`: encuentra una subcadena comenzando por el final de la cadena.
- `rstrip()`: elimina los caracteres en blanco al final de la cadena.
- `split()`: divide la cadena en una subcadena usando un delimitador dado.
- `strip()`: elimina los espacios en blanco iniciales y finales.
- `swapcase()`: intercambia las mayúsculas y minúsculas de las letras.
- `title()`: hace que la primera letra de cada palabra sea mayúscula.
- `upper()`: convierte todas las letras de la cadena en letras mayúsculas.

2. El contenido de las cadenas se puede determinar mediante los siguientes métodos (todos devuelven valores booleanos):

- `endswith()`: ¿La cadena termina con una subcadena determinada?
- `isalnum()`: ¿La cadena consta solo de letras y dígitos?
- `isalpha()`: ¿La cadena consta solo de letras?
- `islower()`: ¿La cadena consta solo de letras minúsculas?
- `isspace()`: ¿La cadena consta solo de espacios en blanco?
- `isupper()`: ¿La cadena consta solo de letras mayúsculas?
- `startswith()`: ¿La cadena consta solo de letras mayúsculas?

Ejercicio 1
¿Cuál es el resultado esperado del siguiente código?

```
for ch in "abc123XYZ":  
    if ch.isupper():  
        print(ch.lower(), end='')  
    elif ch.islower():  
        print(ch.upper(), end='')  
    else:  
        print(ch, end='')
```

ABC123xyz

Ejercicio 2
¿Cuál es el resultado esperado del siguiente código?

```
s1 = '¿Dónde están las nevadas de antaño?'  
s2 = s1.split()  
print(s2[-2])
```

de

PYTHON INSTITUTE

MODULE (45%) SECTION (94%)

Puntos Clave

1. Algunos de los métodos que ofrecen las cadenas son:

- `capitalize()`: cambia todas las letras de la cadena a mayúsculas.
- `center()`: centra la cadena dentro de una longitud conocida.
- `count()`: cuenta las ocurrencias de un carácter dado.
- `join()`: une todos los elementos de una tupla/lista en una cadena.
- `lower()`: convierte todas las letras de la cadena en minúsculas.
- `lstrip()`: elimina los caracteres en blanco al principio de la cadena.
- `replace()`: reemplaza una subcadena dada con otra.
- `rfind()`: encuentra una subcadena comenzando por el final de la cadena.
- `rstrip()`: elimina los caracteres en blanco al final de la cadena.
- `split()`: divide la cadena en una subcadena usando un delimitador dado.
- `strip()`: elimina los espacios en blanco iniciales y finales.
- `swapcase()`: intercambia las mayúsculas y minúsculas de las letras.
- `title()`: hace que la primera letra de cada palabra sea mayúscula.
- `upper()`: convierte todas las letras de la cadena en letras mayúsculas.

2. El contenido de las cadenas se puede determinar mediante los siguientes métodos (todos devuelven valores booleanos):

- `endswith()`: ¿La cadena termina con una subcadena determinada?
- `isalnum()`: ¿La cadena consta solo de letras y dígitos?
- `isalpha()`: ¿La cadena consta solo de letras?
- `islower()`: ¿La cadena consta solo de letras minúsculas?
- `isspace()`: ¿La cadena consta solo de espacios en blanco?
- `isupper()`: ¿La cadena consta solo de letras mayúsculas?
- `startswith()`: ¿La cadena consta solo de letras mayúsculas?

Ejercicio 1
¿Cuál es el resultado esperado del siguiente código?

```
for ch in "abc123XYZ":  
    if ch.isupper():  
        print(ch.lower(), end='')  
    elif ch.islower():  
        print(ch.upper(), end='')  
    else:  
        print(ch, end='')
```

ABC123xyz

Ejercicio 2
¿Cuál es el resultado esperado del siguiente código?

```
s1 = '¿Dónde están las nevadas de antaño?'  
s2 = s1.split()  
print(s2[-2])
```

de

PYTHON INSTITUTE

MODULE (56%) SECTION (93%)

Puntos Claves

1. Las cadenas se pueden comparar con otras cadenas utilizando operadores de comparación generales, pero compararlas con números no da un resultado razonable, porque **ninguna cadena puede ser igual** a ningún otro número. Por ejemplo:

- cadena == número es siempre False (falso).
- cadena != número es siempre True (verdadero).
- cadena >= número siempre genera una excepción.

2. El ordenamiento de listas de cadenas se puede realizar mediante:

- Una función llamada `sorted()`, crea una nueva, lista ordenada.
- Un método llamado `sort()`, el cual ordena la lista en el momento.

3. Un número se puede convertir en una cadena empleando la función `str()`.

4. Una cadena se puede convertir en un número (aunque no todas las cadenas) empleando ya sea la función `int()` o `float()`. La conversión falla si la cadena no contiene un número válido (se genera una excepción en dicho caso).

Ejercicio 1

¿Cuál de las siguientes líneas describe una condición verdadera?

```
1 'smith' > 'Smith'
2 'Smiths' < 'Smith'
3 'Smith' > '1000'
4 '11' < '8'
5
```

Revisar 1, 3 y 4

Ejercicio 2

¿Cuál es el resultado esperado del siguiente código?

```
s1 = '¿Dónde están las nevadas de antaño?'
s2 = s1.split()
s3 = sorted(s2)
print(s3[1])
```

Revisar de

PYTHON INSTITUTE

MODULE (64%) SECTION (100%)

Puntos Claves

1. Las cadenas son herramientas clave en el procesamiento de datos modernos, ya que la mayoría de los datos útiles son en realidad cadenas. Por ejemplo, el uso de un motor de búsqueda web (que parece bastante trivial en estos días) utiliza un procesamiento de cadenas extremadamente complejo, que involucra cantidades inimaginables de datos.

2. El comparar cadenas de forma estricta (como lo hace Python) puede ser muy insatisfactorio cuando se trata de búsquedas avanzadas (por ejemplo, durante consultas extensas a bases de datos). En respuesta a esta demanda, se han creado e implementado una serie de algoritmos de comparación de cadenas *difusos*. Estos algoritmos pueden encontrar cadenas que no son iguales en el sentido de Python, pero que son **similares**.

Uno de esos conceptos es la **Distancia Hamming**, que se utiliza para determinar la similitud de dos cadenas. Si este tema te interesa, puedes encontrar más información al respecto aquí: https://en.wikipedia.org/wiki/Hamming_distance. Otra solución del mismo tipo, pero basada en un supuesto diferente, es la **Distancia Levenshtein** descrita aquí: https://en.wikipedia.org/wiki/Levenshtein_distance.

Ejercicio 3

3. Otra forma de comparar cadenas es encontrar su similitud *acústica*, lo que significa un proceso que lleva a determinar si dos cadenas suenan similares (como "echo" y "hecho"). Esta similitud debe establecerse para cada idioma (o incluso dialecto) por separado. Un algoritmo utilizado para realizar una comparación de este tipo para el idioma Inglés se llama **Soundex** y se inventó, no lo creas, en 1918. Puedes encontrar más información al respecto aquí: <https://en.wikipedia.org/wiki/Soundex>.

4. Debido a la precisión limitada de los datos enteros y flotantes nativos, a veces es razonable almacenar y procesar valores numéricos enormes como cadenas. Esta es la técnica que usa Python cuando se le fuerza a operar con un número entero que consta de una gran cantidad de dígitos.

PYTHON INSTITUTE

MODULE (80%) SECTION (100%)

Punto Clave

1. Una excepción es un evento durante la ejecución del programa causado por una situación anormal. La excepción debe manejarse para evitar la terminación del programa. La parte del código que se sospecha que es la fuente de la excepción debe colocarse dentro del bloque `try`.

Cuando ocurre la excepción, la ejecución del código no se termina, sino que salta al bloque `except`. Este es el lugar donde debe llevarse a cabo el manejo de la excepción. El esquema general para tal construcción es el siguiente:

```
# El código que siempre corre suavemente.
try:
    :
    # Código arriesgado.
    :
except:
    :
    # La gestión de la crisis se lleva a cabo aquí.
    :
# De vuelta a la normalidad.
:
```

2. Si necesitas manejar más de una excepción proveniente del mismo bloque `try`, puedes agregar más de un bloque `except`, pero debes etiquetarlos con diferentes nombres, así:

```
# El código que siempre corre suavemente.
:
```

Ejercicio 1

¿Cuál es el resultado esperado del siguiente código?

```
try:
    print("Tratemos de hacer esto")
    print("#[2]")
    print("Tuvimos éxito!")
except:
    print("Hemos fallado")
    print("Hemos terminado")
```

Revisar Tratemos de hacer esto Hemos fallado Hemos terminado

Ejercicio 2

¿Cuál es el resultado esperado del siguiente código?

```
try:
    print("alpha"[1/0])
except ZeroDivisionError:
    print("cero")
except IndexError:
```

PYTHON INSTITUTE

MODULE (91%) SECTION (100%)

Puntos Clave

1. No se puede agregar más de un bloque `except` sin nombre después de los bloques con nombre.

```
# El código que siempre corre suavemente.
try:
    :
    # Código arriesgado.
    :
except Except_1:
    # La gestión de la crisis se lleva a cabo aquí.
except Except_2:
    # Salvamos el mundo aquí.
except:
    # Todos los demás problemas caen aquí.
    :
    # De vuelta a la normalidad.
    :
```

2. Todas las excepciones de Python predefinidas forman una jerarquía, es decir, algunas de ellas son más generales (la llamada `BaseException` es la más general) mientras que otras son más o menos concretas (por ejemplo, `IndexError` es más concreta que `LookupError`).

Debes evitar colocar excepciones generales antes de las más concretas dentro de la misma secuencia de bloques `except`. Por ejemplo, puedes hacer esto:

Ejercicio 1

¿Cuál es la salida esperada del siguiente código?

```
try:
    print(1/0)
except ZeroDivisionError:
    print("cero")
except ArithmeticError:
    print("arit")
except:
    print("algo")
```

Revisar

cero

Ejercicio 2

¿Cuál es la salida esperada del siguiente código?

```
try:
    print(1/0)
except ArithmeticError:
    print("arit")
except ZeroDivisionError:
```

GUAN

Cerrar sesión

PYTHON INSTITUTE

MODULE (91%) SECTION (83%)

Puntos Clave

1. Algunas excepciones integradas abstractas de Python son:

- `ArithmeticError`,
- `BaseException`,
- `LookupError`.

2. Algunas excepciones integradas concretas de Python son:

- `AssertionError`,
- `ImportError`,
- `IndexError`,
- `KeyboardInterrupt`,
- `KeyError`,
- `MemoryError`,
- `OverflowError`.

Ejercicio 1

¿Cuál de las excepciones se utilizará para proteger al código de ser interrumpido por el uso del teclado?

Revisar

KeyboardInterrupt

Ejercicio 2

¿Cuál es el nombre de la más general de todas las excepciones?

Revisar

BaseException

Ejercicio 3

¿Cuál de las excepciones será generada a través de la siguiente línea de código?

huge_value = 1E250 ** 2

Revisar

PYTHON INSTITUTE

MODULE (11%) SECTION (100%)

Puntos Clave

1. Una **clase** es una idea (más o menos abstracta) que se puede utilizar para crear varias encarnaciones; una encarnación de este tipo se denomina **objeto**.

2. Cuando una clase se deriva de otra clase, su relación se denomina **herencia**. La clase que deriva de la otra clase se denomina **subclase**. El segundo lado de esta relación se denomina **superclase**. Una forma de presentar dicha relación es en un **diagrama de herencia**, donde:

- Las superclases siempre se presentan **encima** de sus subclases.
- Las relaciones entre clases se muestran como flechas dirigidas **desde la subclase hacia su superclass**.

3. Los objetos están equipados con:

- Un **nombre** que los identifica y nos permite distinguirlos.
- Un conjunto de **propiedades** (el conjunto puede estar vacío).
- Un conjunto de **métodos** (también puede estar vacío).

4. Para definir una clase de Python, se necesita usar la palabra clave reservada `class`. Por ejemplo:

```
class This_Is_A_Class:
    pass
```

5. Para crear un objeto de la clase previamente definida, se necesita usar la clase como si fuera una función. Por ejemplo:

Ejercicio 1

Si asumimos que pitones, víboras y cobras son subclases de la misma superclass, ¿cómo la llamarías?

Revisar

Serpiente, reptil, vertebrado, animal: todas estas respuestas son aceptables.

Ejercicio 2

Intenta nombrar algunas subclases de la clase Pitón.

Revisar

Pitón India, Pitón de Roca Sificana, Pitón Bola, Pitón Birmano

Ejercicio 3

¿Puedes usar la palabra "class" para darle nombre a algún objeto?

Revisar

(No, no puedes, class es una palabra clave reservada)

GUAN

Cerrar sesión

PYTHON INSTITUTE

MODULE (26%) SECTION (81%)

Puntos Clave

- Una **pila** es un objeto diseñado para almacenar datos utilizando el modelo **LIFO**. La pila normalmente realiza al menos dos operaciones, llamadas **push()** y **pop()**.
- La implementación de la pila en un modelo procedural plantea varios problemas que pueden resolverse con las técnicas ofrecidas por la **POO** (Programación Orientada a Objetos).
- Un **método de clase** es en realidad una función declarada dentro de la clase y capaz de acceder a todos los componentes de la clase.
- La parte de la clase en Python responsable de crear nuevos objetos se llama **constructor** y se implementa como un método de nombre `__init__`.
- Cada declaración de método de clase debe contener al menos un parámetro (siempre el primero) generalmente denominado `self`, y es utilizado por los objetos para identificarse a sí mismos.
- Si queremos ocultar alguno de los componentes de una clase del mundo exterior, debemos comenzar su nombre con `_`. Estos componentes se denominan **privados**.

Ejercicio 1

Suponiendo que hay una clase llamada `Snakes`, escribe la primera línea de la declaración de clase `Python`, expresando el hecho de que la nueva clase es en realidad una subclase de `Snake`.

Revisar

```
class Python(Snakes):
```

Ejercicio 2

Algo falta en la siguiente declaración, ¿qué es?

```
class Snakes:
    def __init__():
        self.sound = 'Sssssss'
```

Revisar

El constructor `__init__()` carece del parámetro obligatorio `self`.

Ejercicio 3

PYTHON INSTITUTE

MODULE (40%) SECTION (100%)

Puntos Clave

- Una **variable de instancia** es una propiedad cuya existencia depende de la creación de un objeto. Cada objeto puede tener un conjunto diferente de variables de instancia.
- Además, se pueden agregar y quitar libremente de los objetos durante su vida útil. Todas las variables de instancia de objeto se almacenan dentro de un diccionario dedicado llamado `__dict__`, contenido en cada objeto por separado.
- Una variable de instancia puede ser privada cuando su nombre comienza con `_`, pero no olvides que dicha propiedad aún es accesible desde fuera de la clase usando un **nombre modificado** construido como `<classname>_PropertyName`.
- Una **variable de clase** es una propiedad que existe exactamente en una copia y no necesita ningún objeto creado para ser accesible. Estas variables no se muestran como contenido de `__dict__`.
- Todas las variables de clase de una clase se almacenan dentro de un diccionario dedicado llamado `__dict__`, contenido en cada clase por separado.
- Una función llamada `hasattr()` se puede utilizar para determinar si algún objeto o clase contiene cierta propiedad especificada.

Por ejemplo:

```
1 class Sample:
2     gamma = 0 # Class variable.
3     def __init__(self):
4         self.alpha = 1 # Variable de instancia.
5         self.__delta = 3 # Variable de instancia privada.
```

Ejercicio 1

¿Cuáles de las propiedades de la clase `Python` son variables de instancia y cuáles son variables de clase? ¿Cuáles de ellos son privados?

```
class Python:
    population = 1
    victims = 0
    def __init__(self):
        self.length_ft = 3
        self.__venomous = False
```

Revisar

`population` y `victims` son variables de clase, mientras que `length_ft` y `__venomous` son variables de instancia y también es **privada**.

Ejercicio 2

Vas a negar la propiedad `__venomous` del objeto `versus` haciendo esto?

```
version_2 = Python()
```

Revisar

PYTHON INSTITUTE

MODULE (54%) SECTION (73%)

Puntos Clave

- Un método es una función dentro de una clase. El primer (o único) parámetro de cada método se suele llamar `self`, que está diseñado para identificar al objeto para el que se invoca el método con el fin de acceder a las propiedades del objeto o invocar sus métodos.
- Si una clase contiene un **constructor** (un método llamado `__init__`), este no puede devolver ningún valor y no se puede invocar directamente.
- Todas las clases (pero no los objetos) contienen una propiedad llamada `__name__`, que almacena el nombre de la clase. Además, una propiedad llamada `__module__` almacena el nombre del módulo en el que se ha declarado la clase, mientras que la propiedad llamada `__bases__` es una tupla que contiene las superclases de una clase.

Por ejemplo:

```
1 class Sample:
2     def __init__(self):
3         self.name = Sample.__name__
4         self.myself(self):
5             print("Mi nombre es " + self.name + " y vivo en " + Sample.__module__)
6
7
8 obj = Sample()
9 obj.myself()
```

El código da como salida:

```
My nombre es Sample y vivo en __main__
```

Ejercicio 1

La declaración de la clase `Snake` se muestra a continuación. Enriquece la clase con un método llamado `increment()`, el cual incrementa en 1 la propiedad `victims`.

```
class Snake:
    def __init__(self):
        self.victims = 0
```

Revisar

```
class Snake:
    def __init__(self):
        self.victims = 0

    def increment(self):
        self.victims += 1
```

Ejercicio 2

Redefine el constructor de la clase `Snake` para que tenga un argumento durante la construcción.

Revisar

Puntos Clave

- Un método llamado `__str__()` es responsable de convertir el contenido de un objeto en una cadena (más o menos) legible. Puedes redefinirlo si deseas que tu objeto pueda presentarse de una forma más elegante. Por ejemplo:

```
1 class Mouse:
2     def __init__(self, name):
3         self.my_name = name
4
5     def __str__(self):
6         return self.my_name
7
8
9
10 the_mouse = Mouse('mickey')
11 print(the_mouse) # Imprime "mickey".
12
```

- Una función llamada `issubclass(Class_1, Class_2)` es capaz de determinar si `Class_1` es una **subclase** de `Class_2`. Por ejemplo:

```
1 class Mouse:
2     pass
3
4
5 class LabMouse(Mouse):
6     pass
7
8
9 print(issubclass(Mouse, LabMouse), issubclass(LabMouse, Mouse)) # Imprime "False True"
10
```

- Una función llamada `isinstance(Object, Class)` comproueba si un objeto proviene de una clase indicada. Por ejemplo:

5. Una función sin parámetros llamada `super()` retorna la **referencia a la superclase más cercana de la clase**. Por ejemplo:

```
1 class Mouse:
2     def __init__(self):
3         return "Mouse"
4
5
6 class LabMouse(Mouse):
7     def __str__(self):
8         return "laboratory" + super().__str__()
9
10 doctor_mouse = LabMouse()
11 print(doctor_mouse) # Imprime "laboratoryMouse"
```

6. Los métodos, así como las variables de instancia y de clase, tienen su propia jerarquía de herencia. Por ejemplo:

```
1 class Mouse:
2     Population = 0
3     def __init__(self, name):
4         Mouse.Population += 1
5         self.name = name
6
7     def __str__(self):
8         return "Hola, mi nombre es " + self.name
9
10 class LabMouse(Mouse):
11     pass
12
13 professor_mouse = LabMouse("Profesor Mou")
14 print(professor_mouse, Mouse.Population)
```

Ejercicios

Escenario

El siguiente fragmento de código se ha ejecutado con éxito:

```
1 class Dog:
2     kennel = 0
3     def __init__(self, breed):
4         self.breed = breed
5         Dog.kennel += 1
6     def __str__(self):
7         return self.breed + " dice: ¡Guau!"
8
9
10 class SheepDog(Dog):
11     def __str__(self):
12         return super().__str__() + " ¡No huyas, corderito!"
13
14
15 class GuardDog(Dog):
16     def __str__(self):
17         return super().__str__() + " ¡Quédese donde está, intruso!"
18
19
20 rocky = SheepDog("collie")
21 luna = GuardDog("dobermann")
22
```

Ahora responde las preguntas de los ejercicios 1-4.

Ejercicio 1

¿Cuál es el resultado esperado del siguiente código?

```
print(rocks)
print(luna)
```

Ejercicio 2

¿Cuál es el resultado esperado del siguiente código?

```
print(issubclass(SheepDog, Dog), issubclass(
    print(isinstance(rocks, GuardDog), isinstance(luna, SheepDog)))
```

Puntos Clave

- El bloque `else:` de la sentencia `try` se ejecuta cuando no ha habido ninguna excepción durante la ejecución del `try`:
- El bloque `finally:` de la sentencia `try` es **siempre** ejecutado.
- La sintaxis `except Exception_Name as exception_object:` te permite interceptar un objeto que contiene información sobre una excepción pendiente. La propiedad del objeto llamada `args` (una tupla) almacena todos los argumentos pasados al constructor del objeto.
- Las clases de excepciones pueden extenderse para enriquecerlas con nuevas capacidades o para adoptar sus características excepciones recién definidas.

Por ejemplo:

```
try:
    assert __name__ == "__main__"
except:
    print("fallido", end=' ')
else:
    print("éxito", end=' ')
finally:
    print("terminado")
```

El código da como salida: éxito terminado.

Ejercicio 1

¿Cuál es el resultado esperado del siguiente código?

```
import math

try:
    print(math.sqrt(9))
except ValueError:
    print("inf")
else:
    print("ok")
```

Ejercicio 2

¿Cuál es el resultado esperado del siguiente código?

```
import math

try:
    print(math.sqrt(-9))
except ValueError:
    print("inf")
```

PYTHON INSTITUTE

MODULE (10%) SECTION (100%)

Puntos Clave

- Un **iterador** es un objeto de una clase que proporciona al menos **dos** métodos (sin contar el constructor):
 - `__iter__()` se invoca una vez cuando se crea el iterador y devuelve el **propio** objeto del iterador.
 - `__next__()` se invoca para proporcionar el **valor de la siguiente iteración** y genera la excepción `StopIteration` cuando la iteración **llega a su fin**.
- La sentencia `yield` solo puede ser utilizada dentro de funciones. La sentencia `yield` suspende la ejecución de la función y hace que la función regrese el argumento de `yield` como resultado. Esta función no puede invocarse de forma regular, su único propósito es ser utilizada como un **generador** (es decir, en un contexto que requiera una serie de valores, como un bucle `for`).
- Una **expresión condicional** es una expresión construida usando el operador `if-else`. Por ejemplo:

```
print(True if 0 == 0 else False)
```

Da como salida: True .

- Una **lista por comprensión** se convierte en un **generador** cuando se emplea dentro de **paréntesis** (usado entre corchetes, produce una lista regular). Por ejemplo:

```
for x in (el * 2 for el in range(5)):
    print(x)
```

Da como salida: 02468 .

Ejercicio 1

¿Cuál es el resultado esperado del siguiente código?

```
class Vowels:
    def __init__(self):
        self.vow = "aeiouy" # Si, sabemos que y no siempre se considera una vocal.
        self.pos = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.pos == len(self.vow):
            raise StopIteration
        self.pos += 1
        return self.vow[self.pos - 1]

vowels = Vowels()
for v in vowels:
    print(v, end=" ")
```

Check

a e i o u y

Ejercicio 2

Universidad Tecnológica de Panamá Plataforma eLearn... AXEL ALEJANDRO ALMAGUER RODRÍGUEZ Área personal Perfil Calificaciones Mensajes Preferencias GUAN Cerrar sesión

PYTHON INSTITUTE

MODULE (30%) SECTION (100%)

Puntos Clave

- Un archivo necesita ser **abierto** antes de que pueda ser procesado por un programa, y debe ser **cerrado** cuando el procesamiento termine.
- El abrir un archivo lo asocia con el **stream**, que es una representación abstracta de los datos físicos almacenados en los medios. La forma en que se procesa el stream se llama **modo de apertura**. Existen **tres** modos de apertura:
 - modo lectura**: solo se permiten operaciones de lectura.
 - modo escritura**: solo se permiten operaciones de escritura.
 - modo de actualización**: se permiten ambas, lectura y escritura.
- Dependiendo del contenido del archivo físico, se pueden usar diferentes clases de Python para procesar archivos. En general, `BufferedIOBase` es capaz de procesar cualquier archivo, mientras que `TextIOWrapper` es una clase especializada dedicada al procesamiento de archivos de texto (es decir, archivos que contienen textos visibles para humanos divididos en líneas usando marcadores de nueva línea). Por lo tanto, los streams se pueden dividir en **binarios** y **de texto**.
- Las siguientes sintaxis de la función `open()` se utilizan para abrir un archivo:

```
open(nombre_archivo, modo=modo_apertura, codificación=codificación_de_texto)
```

La invocación crea un objeto `stream` y lo asocia con el archivo llamado `nombre_archivo`, utilizando el modo `modo_apertura` y configurando la especificada `codificación_de_texto`, o **genera una excepción en caso de un error**.

- Los tres streams **predefinidos** que ya están abiertos cuando inicia el programa son:
 - `sys.stdin` à entrada estandar.

Ejercicio 1

¿Cómo se codifica el valor del argumento modo de la función `open()` si se va a crear un nuevo archivo de texto?

Revisar

"wt" o "w"

Ejercicio 2

¿Cuál es el significado del valor representado por `errno`?

Revisar

Ejercicio 3

Permito denegado: no se permite acceder al contenido de un archivo.

Universidad Tecnológica de Panamá Plataforma eLearn... AXEL ALEJANDRO ALMAGUER RODRÍGUEZ Área personal Perfil Calificaciones Mensajes Preferencias GUAN Cerrar sesión

PYTHON INSTITUTE

MODULE (40%) SECTION (100%)

Puntos Clave

- Para leer el contenido de un archivo, se pueden utilizar los siguientes métodos:
 - `read(number)`: lee el `número` de caracteres/bytes del archivo y lo retorna como una cadena, es capaz de leer todo el archivo a la vez.
 - `readline()`: lee una sola línea del archivo de texto.
 - `readlines(number)`: lee el `número` de líneas del archivo de texto; es capaz de leer todas las líneas a la vez.
 - `readinto(bytarray)`: lee los bytes del archivo y llena el `bytarray` con ellos.
- Para escribir contenido nuevo en un archivo, se pueden utilizar los siguientes métodos:
 - `write(string)`: escribe una cadena a un archivo de texto.
 - `write(bytarray)`: escribe todos los bytes de un `bytarray` a un archivo.
- El método `open()` devuelve un objeto iterable que se puede usar para recorrer todas las líneas del archivo dentro de un bucle `for`. Por ejemplo:

```
for line in open("file", "rt"):
    print(line, end="")
```

El código copia el contenido del archivo a la consola, línea por línea. **Nota:** el stream se cierra **automáticamente** cuando llega al final del archivo.

Ejercicio 1

¿Qué se espera del método `readlines()` cuando el stream está asociado con un archivo vacío?

Revisar

Una lista vacía (una lista de longitud cero).

Ejercicio 2

¿Qué se pretende hacer con el siguiente código?

```
for line in open("file", "rt"):
    for char in line:
        if char.lower() not in "aeiouy":
            print(char, end="")
```

Revisar

Copia el contenido del archivo `file` hacia la consola, ignorando las vocales.

Ejercicio 3

Vas a procesar un mapa de bits almacenado en un archivo variable `bitsarr`. Llama a `imape` para leer una línea al d

Universidad Tecnológica de Panamá Plataforma eLearn... AXEL ALEJANDRO ALMAGUER RODRÍGUEZ Área personal Perfil Calificaciones Mensajes Preferencias GUAN Cerrar sesión

PYTHON INSTITUTE
Open Education & Development Corp.

MODULE (50%) SECTION (100%)

Puntos Claves

- La función `uname` devuelve un objeto que contiene información sobre el sistema operativo actual. El objeto tiene los siguientes atributos:
 - `systemname` (almacena el nombre del sistema operativo)
 - `nodename` (almacena el nombre de la máquina en la red)
 - `releasename` (almacena el release (actualización) del sistema operativo)
 - `version` (almacena la versión del sistema operativo)
 - `machine` (almacena el identificador de hardware, por ejemplo, x86_64)
- El atributo `name` disponible en el módulo `os` te permite distinguir el sistema operativo. Devuelve uno de los siguientes tres valores:
 - `posix` (obtendrás este nombre si usas Unix)
 - `nt` (obtendrás este nombre si usas Windows)
 - `java` (obtendrá este nombre si tu código está escrito en algo como python)
- La función `mkdir` crea un directorio en la ruta pasada como argumento. La ruta puede ser relativa o absoluta, por ejemplo:

```
import os
os.mkdir("hello") # la ruta relativa
os.mkdir("/home/python/hello") # la ruta absoluta
```

Nota: Si el directorio existe, una excepción `FileNotFoundError` será generada. Además de la función `mkdir`, el módulo `os` proporciona la función `makedirs`, que te permite crear recursivamente todos los directorios en una ruta.

- El resultado de la función `listdir()` es una lista que contiene los nombres de los archivos y directorios que se encuentran en la ruta pasada como argumento.

Ejercicio 1
¿Cuál es el resultado del siguiente fragmento si se ejecuta en Unix?

```
import os
print(os.name)
```

Ejercicio 2
¿Cuál es el resultado del siguiente fragmento de código?

```
import os
os.mkdir("hello")
print(os.listdir())
```

[`'hello'`]

PYTHON INSTITUTE
Open Education & Development Corp.

MODULE (50%) SECTION (100%)

Punto Clave

- Para crear un objeto `date`, debes pasar los argumentos de año, mes y día de la siguiente manera:

```
from datetime import date
my_date = date(2020, 9, 29)
print("Año:", my_date.year) # Año: 2020
print("Mes:", my_date.month) # Mes: 9
print("Día:", my_date.day) # Día: 29
```

El objeto `date` tiene tres atributos (de solo lectura): año, mes y día.

- El método `today` devuelve un objeto de fecha que representa la fecha local actual:

```
from datetime import date
print("Hoy:", date.today()) # Muestra: Hoy: 2020-09-29
```

En Unix, la marca de tiempo expresa el número de segundos desde el 1 de Enero de 1970 a las 00:00:00 (UTC). Esta fecha se llama la "época de Unix", porque ahí comenzó el conteo del tiempo en los sistemas Unix. La marca de tiempo es en realidad la diferencia entre una fecha en particular (incluida la hora) y el 1 de Enero de 1970, 00:00:00 (UTC), expresada en segundos. Para crear un objeto de fecha a partir de una marca de tiempo, debemos pasar una marca de tiempo Unix al método `fromtimestamp`:

```
from datetime import date
import time
timestamp = time.time()
d = date.fromtimestamp(timestamp)
```

Ejercicio 1
¿Cuál es el resultado del siguiente fragmento de código?

```
from datetime import time
t = time(14, 39)
print(t.strftime("%H:%M:%S"))
```

14:53:00

Ejercicio 2
¿Cuál es el resultado del siguiente fragmento de código?

```
from datetime import datetime
dt1 = datetime(2020, 9, 29, 14, 41, 0)
dt2 = datetime(2020, 9, 28, 14, 41, 0)
print(dt1 - dt2)
```

Revise

PYTHON INSTITUTE
Open Education & Development Corp.

MODULE (70%) SECTION (90%)

Puntos Claves

- En el módulo `calendar`, los días de la semana se muestran de Lunes a Domingo. Cada día de la semana tiene su representación en forma de número entero, donde el primer día de la semana (Lunes) está representado por el valor 0, mientras que el último día de la semana (Domingo) está representado por el valor 6.
- Para mostrar un calendario de cualquier año, se emplea la función `calendar` con el año pasado como argumento, por ejemplo:

```
import calendar
print(calendar.calendar(2020))
```

Nota: Una buena alternativa a la función anterior es la función llamada `prcal`, que también toma los mismos parámetros que la función `calendar`, pero no requiere el uso de la función `print` para mostrar el calendario.

- Para mostrar un calendario de cualquier mes del año, se emplea la función `month`, pasándole el año y el mes. Por ejemplo:

```
import calendar
print(calendar.month(2020, 9))
```

Nota: También puedes usar la función `prmonth`, que tiene los mismos parámetros que la función `month`, pero no requiere el uso de la función `print` para mostrar el calendario.

- La función `setfirstweekday` te permite cambiar el primer día de la semana. Toma un valor de 0 a 6, donde 0 es Domingo y 6 es Sábado.

Ejercicio 1
¿Cuál es el resultado del siguiente fragmento de código?

```
import calendar
print(calendar.weekheader(1))
```

Ejercicio 2
¿Cuál es el resultado del siguiente fragmento de código?

```
import calendar
c = calendar.Calendar()
for weekday in c.iterweekdays():
    print(weekday, end=" ")
```

Revise