

El alumnado crea archivo en formato PDF donde se evidencia respuestas a cada ejercicio de todas las secciones RESUMEN DE LA SECCION

PYTHON INSTITUTE  
Institutional Accredited

MODULE (20%)

SECTION (100%)

<>⌵⚙️🔍

1.1.1.1.1 RESUMEN DE SECCIÓN >

My Profile | Networking Academy

Cristopher López Jiménez  
Networking Academy ID : 1041497772  
Email : szzermin@gmail.com [Change email](#)

<=>🔄🏠🔒📄🔗🔍📎🗑️☰

## Puntos Claves

1. 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
```

2. Si un módulo se importa de la manera anterior y desea 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.

3. 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 línea:

```
import mint
```

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

Revisar

```
mint.make_money()
```

### Ejercicio 2

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

```
from mint import make_money
```

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

Revisar

```
make_money()
```

Python Institute

1.2.1.17 RESUMEN DE SECCIÓN

MODULE (48%)SECTION (100%)

## Puntos Clave

- Una función llamada `dir()` puede mostrarte 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 `pi` 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)
```

Revisar

True

Ejercicio 2

(Completa el enunciado) Establecer la semilla del generador con el mismo valor cada vez que se ejecuta tu programa garantiza que ...

Revisar

... los valores pseudoaleatorios emitidos desde el módulo `random` serán exactamente los mismos.

Ejercicio 3

¿Cuál de las funciones del módulo `platform` utilizarías para determinar el nombre del CPU que corre dentro de tu computadora?

Revisar

La función `processor()`

**PYTHON INSTITUTE**  
ANALYTIC POWER

MODULE (57%)

« 1.3.1.11 RESUMEN DE SECCIÓN »

SECTION (100%)

≡
🔍
⚙️
<>

My Profile | Networking Academy

Christopher López Jiménez

Networking Academy ID: 1041497772

Email: azzerrin@gmail.com [Change email](#)

### Puntos Clave

- Mientras que un **módulo** está diseñado para acoplar 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 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

Desear evitar que el usuario de tu módulo ejecute tu código como un script. ¿Cómo lo logras tal efecto?

Pista

```
import sys

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

### Ejercicio 2


Algunos paquetes adicionales y necesarios se almacenan dentro del directorio `D:\Python\Project\Modules`. Escribe un código asegurándote de que Python recorra el directorio para encontrar todos los módulos solicitados.

Pista

```
import sys

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

Cristopher López Jiménez  
Networking Academy ID: 1041497772  
Email: [azzermin@gmail.com](mailto:azzermin@gmail.com) [Change email](#)



My Profile | Networking Academy

Cristopher López Jiménez

Networking Academy ID: 1041497772

Email: azzermin@gmail.com [Change email](#)

Cristopher López Jiménez

Networking Academy ID: 1041497772  
Email: [azzermin@gmail.com](mailto:azzermin@gmail.com) [Change email](#)

## 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='')
```

Revisar

ABC123xyz

### Ejercicio 2

¿Cuál es el resultado esperado del siguiente código?

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

Revisar

de

Cristopher López Jiménez

Networking Academy ID: 1041497772  
Email: azztermin@gmail.com [Change email](#)

## 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?

### 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='')
```

Revisar

ABC123xyz

### Ejercicio 2

¿Cuál es el resultado esperado del siguiente código?

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

Cristopher López Jiménez

Networking Academy ID: 1041497772  
Email: azztermin@gmail.com [Change email](#)

## 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 'Smiths' > '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

Cristopher López Jiménez

Networking Academy ID: 1041497772  
Email: azztermin@gmail.com [Change email](#)

PYTHON

INSTITUTE

DE LA UNIVERSIDAD DE CORDOBA

2.5.1.5 RESUMEN DE SECCIÓN

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](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](https://en.wikipedia.org/wiki/Levenshtein_distance).

3. Otra forma de comparar cadenas es encontrar su longitud. Las cadenas que tienen la misma longitud pero diferentes caracteres suenan similares (como "echo" y "hecho").

Un algoritmo utilizado para realizar una comparación de este tipo para el idioma Inglés se llama **Soundex** y se inventó, no lo creáis, 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.

My Profile | Networking Academy

Cristopher López Jiménez

Networking Academy ID: 1041497772

Email: [azzermin@gmail.com](mailto:azzermin@gmail.com) [Change email](#)

PYTHON

INSTITUTE

DE LA UNIVERSIDAD DE CORDOBA

2.6.1.12 RESUMEN DE SECCIÓN

MODULE (64%)

SECTION (100%)

### Puntos 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:
    print("cero")
```

My Profile | Networking Academy

Cristopher López Jiménez

Networking Academy ID: 1041497772

Email: [azzermin@gmail.com](mailto:azzermin@gmail.com) [Change email](#)

PYTHON

INSTITUTE

DE LA UNIVERSIDAD DE CORDOBA

2.7.1.8 RESUMEN DE SECCIÓN

MODULE (64%)

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:
```

My Profile | Networking Academy

Cristopher López Jiménez

Networking Academy ID: 1041497772

Email: [azzermin@gmail.com](mailto:azzermin@gmail.com) [Change email](#)

PYTHON INSTITUTE

Python for Professionals

2.8.1.5 RESUMEN DE SECCIÓN

MODULE (97%)

SECTION (93%)

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 un uso del teclado?

Revisar

KeyboardInterrupt

Ejercicio 2

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

Revisar

BaseException

Ejercicio 3

¿Cuál de las excepciones será generada a través de la siguiente evaluación fallida?

Revisar

huge\_value = 1E250 \*\* 2

My Profile | Networking Academy

Cristopher López Jiménez

Networking Academy ID: 1041497772

Email: azzzermin@gmail.com [Change email](#)

PYTHON INSTITUTE

Python for Professionals

3.1.1.8 RESUMEN DE SECCIÓN

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 superclase**.

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 superclase, ¿cómo la llamarías?

Revisar

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

Ejercicio 2

Intenta nombrar algunas subclases de las clase Pitón.

Revisar

Pitón india, Pitón de Roca Sfricana, Pitón Bola, Pitón Birmana: la lista es larga.

Ejercicio 3

¿Puedes usar la palabra "class" para darle nombre a alguna de tus clases?

Revisar

¡No, no puedes, `class` es una palabra clave reservada!

My Profile | Networking Academy

Cristopher López Jiménez

Networking Academy ID: 1041497772

Email: azzzermin@gmail.com [Change email](#)

PYTHON INSTITUTE

Python for Professionals

3.2.1.13 RESUMEN DE SECCIÓN

MODULE (26%)

SECTION (91%)

Puntos Clave

1. 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()**.

2. 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)**.

3. 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.

4. La parte de la clase en Python responsable de crear nuevos objetos se llama **constructor** y se implementa como un método de nombre `__init__` .

5. 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.

6. 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** de 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?

Revisar

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

El constructor `__init__()` carece del parámetro obligatorio (deberíamos llamarlo `self` para cumplir con los estándares).

Ejercicio 3

Modifica el código para garantizar que la propiedad `venomous` sea privada.

My Profile | Networking Academy

Cristopher López Jiménez

Networking Academy ID: 1041497772

Email: azzzermin@gmail.com [Change email](#)

Cristopher López Jiménez

Networking Academy ID: 1041497772  
Email: azzarmin@gmail.com [Change email](#)

Cristopher López Jiménez

Networking Academy ID: 1041497772

Email: [azzerman@gmail.com](mailto:azzerman@gmail.com) [Change email](#)

Cristopher López Jiménez

Networking Academy ID: 1041497772  
Email: [azzermin@gmail.com](mailto:azzermin@gmail.com) [Change email](#)

Python Institute

Professional Education

3.5.1.23 RESUMEN DE SECCIÓN 2/2

MODULE (27%)

SECTION (100%)

My Profile | Networking Academy

← → ↺ 🔒 http:// ... ⌵ ☰

Cristopher López Jiménez

Networking Academy ID: 1041497772

Email: azzemini@gmail.com [Change email](#)

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("Doberman")
22
```

Ahora responde las preguntas de los ejercicios 1-4.

Ejercicio 1

¿Cuál es el resultado esperado del siguiente código?

```
print(rocky)
print(luna)
```

Revisar

Collie dice: ¡Guau! ¡No huyas, corderito!  
Doberman dice: ¡Guau! ¡Quédese donde está, intruso!

Ejercicio 2

¿Cuál es el resultado esperado del siguiente código?

```
print(isinstance(SheepDog, Dog), isinstance(SheepDog, GuardDog))
print(isinstance(rocky, GuardDog), isinstance(luna, GuardDog))
```

Revisar

True False  
False True

Python Institute

Professional Education

3.6.1.9 RESUMEN DE SECCIÓN

MODULE (76%)

SECTION (100%)

My Profile | Networking Academy

← → ↺ 🔒 http:// ... ⌵ ☰

Cristopher López Jiménez

Networking Academy ID: 1041497772

Email: azzemini@gmail.com [Change email](#)

Puntos Clave

1. El bloque `else:` de la sentencia `try` se ejecuta cuando no ha habido ninguna excepción durante la ejecución del `try`.

2. El bloque `finally:` de la sentencia `try` es **siempre** ejecutado.

3. 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.

4. Las clases de excepciones pueden extenderse para enriquecerlas con nuevas capacidades o para adoptar sus características a 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")
```

Revisar

3.0  
ok

Ejercicio 2

¿Cuál es el resultado esperado del siguiente código?

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

Python Institute

Professional Education

4.1.1.15 RESUMEN DE SECCIÓN

MODULE (77%)

SECTION (100%)

My Profile | Networking Academy

← → ↺ 🔒 http:// ... ⌵ ☰

Cristopher López Jiménez

Networking Academy ID: 1041497772

Email: azzemini@gmail.com [Change email](#)

Puntos Clave

1. 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**.

2. 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`).

3. 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.

4. 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]

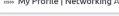
vowels = Vowels()
for v in vowels:
    print(v, end=' ')
```

Check

a e i o u y

Ejercicio 2

A screenshot of a web browser window. The address bar shows "My Profile | Networking Ac...". The page content displays the name "Cristopher López Jiménez" in a large font. Below the name, it says "Networking Academy ID: 1041497772" and "Email: azzermin@gmail.com Change email".



Christopher López Jiménez

Networking Academy ID: 1041497772

Email: azzermin@gmail.com [Change email](#)

A screenshot of a web browser window. The address bar shows "My Profile | Networking Ac...". The main content area displays the name "Christopher López Jiménez" in large black font. Below it, in smaller grey font, are the details: "Networking Academy ID: 1041497772", "Email: azzermin@gmail.com", and a link "Change email" in blue. A small portion of another tab titled "P..." is visible at the bottom left.



Python Institute

Professional Networking Academy

4.5.1.23 RESUMEN DE SECCIÓN

MODULE (33%)

SECTION (100%)

≡

🔍

⚙️

📄

### Puntos Clave

1. 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.

2. 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
```

3. 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
```

#### 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"))
```

Revisar

14:53:00

My Profile | Networking Academy

+

▼

Cristopher López Jiménez

Networking Academy ID: 1041497772

Email: [azzermin@gmail.com](mailto:azzermin@gmail.com) [Change email](#)

Python Institute

Professional Networking Academy

4.6.1.14 RESUMEN DE SECCIÓN

MODULE (37%)

SECTION (33%)

≡

🔍

⚙️

📄

### Puntos Claves

1. 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.

2. 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.

3. 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.

4. 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))
```

Revisar

M T W T F S S

#### Ejercicio 2

¿Cuál es el resultado del siguiente fragmento de código?

```
import calendar

c = calendar.Calendar()

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

Revisar

0 1 2 3 4 5 6

My Profile | Networking Academy

+

▼

Cristopher López Jiménez

Networking Academy ID: 1041497772

Email: [azzermin@gmail.com](mailto:azzermin@gmail.com) [Change email](#)