Comenzado el	sábado, 16 de junio de 2018, 17:59
Estado	Finalizado
Finalizado en	sábado, 16 de junio de 2018, 18:38
Tiempo empleado	39 minutos 44 segundos
Puntos	13/13
Calificación	**10** de 10 (**100**%)

Pregunta 1

Correcta

Puntúa 1 sobre 1

Suponga la siguiente cabecera de una función cuyo bloque de acciones es irrelevante a los efectos de la pregunta:

```
def ejemplo(a, b=0, c='Desconocido', d='Ingeniero'):
    # el bloque de acciones no importa ahora...
```

¿Cúales de las siguientes invocaciones a esta función son correctas y activan la función sin producir un error de intérprete? (Observación: más de una respuesta puede ser correcta. Marque todas las que considere válidas... y tómese su tiempo para pensar y probar cada respuesta posible...)

Seleccione una o más de una:

- [x] a.
`ejemplo('Medico')` ✓

  Correcto. Efectivamente, un solo parámetro es obligatorio en esta función y por lo tanto el único valor enviado ('Medico') se asigna en ese parámetro (la variable *a*). El resto toma sus valores por default *[Valores finales de los parámetros: a = Medico, b = 0, c = "Desconocido", d = "Ingeniero"] Si este resultado le parece extaño, ¡recuerde que los parámetros se toman y se asignan por orden de aparición, y que Pyton es un lenguaje de tipado dinámico!*

- [ ] b.
`ejemplo()`

- [x] c.
`ejemplo(32, 20)` ✓

  Correcto. Efectivamente, el primer parámetro es obligatorio (y en este caso es un número), y el segundo es opcional dado que admite un valor default. Los otros dos quedan valiendo valores default *[Valores finales de los parámetros: a = 32, b = 20, c = "Desconocido", d = "Ingeniero"]*

- [x] d.
`ejemplo(40, 'Juan', 'Abogado')` ✓

  Correcto. Efectivamente, el primer parámetro es obligatorio (y en este caso es un número). El segundo y el tercero quedan valiendo las cadenas "Juan" y "Abogado". Y el restante queda valiendo su valor default *[Valores finales de los parámetros: a = 40, b = "Juan", c = "Abogado", d = "Ingeniero"] Si este resultado le parece extaño, ¡recuerde que los parámetros se toman y se asignan por orden de aparición, y que Python es un lenguaje de tipado dinámico!*

¡Correcto!

Las respuestas correctas son:
`ejemplo(32, 20)`,

`ejemplo(40, 'Juan', 'Abogado')`,

```
ejemplo('Medico')
```

Pregunta 2

Correcta

Puntúa 2 sobre 2

Suponga que se quiere desarrollar una función que tome como parámetro una secuencia (un string, una lista, una tupla, o cualquier otro tipo de colección que permita acceso mediante índices en Python) y que proceda a ordenar esa colección con diversos algoritmos conocidos, en forma ascendente o descendente a elección de quien invoca a la función.

En ese contexto, considere la siguiente definición para esa función, en la cual no es relevante su bloque de acciones a los efectos de la pregunta:

```
def ordenar(lista, ascendente=True, algoritmo='Quicksort'):
    # el bloque de acciones no importa ahora...
```

¿Cuáles de las siguientes invocaciones a esta función son correctas, y no provocarán un error de intérprete? (Observación: más de una respuesta puede ser válida. Marque todas las que considere adecuadas... y de nuevo: tómese su tiempo!!!)

Seleccione una o más de una:

- [x] a.
`ordenar('azbycx', algoritmo='Shellsort')` ✓

  Correcto. Efectivamente, el primer parámetro es obligatorio en esta función. El segundo queda con valor default. Y el tercero es seleccionado por palabra clave [Valores finales de los parámetros: lista = "azbycx", ascendente = True, algoritmo = "Shellsort"]

- [ ] b.
`ordenar('abcdef', ascendente=False, 'Insertionsort')`

- [ ] c.
`ordenar('abcdef', False, metodo='Bubblesort')`

- [x] d.
`ordenar(lista=(1,2,3), algoritmo='Heapsort', ascendente=False)` ✓

  Correcto. Efectivamente, el primer parámetro es obligatorio en esta función, y no sólo eso sino que en este caso se lo está accediendo por palabra clave y queda valiendo una tupla compuesta de números. El segundo y el tercero son seleccionados por palabra clave, sin importar si se los está accediendo en orden diferente al de su declaración. [Valores finales de los parámetros: lista = (1,2,3), ascendente = False, algoritmo = "Heapsort"]

¡Correcto!

Las respuestas correctas son:
`ordenar('azbycx', algoritmo='Shellsort')`,

`ordenar(lista=(1,2,3), algoritmo='Heapsort', ascendente=False)`

Pregunta 3

Correcta

Puntúa 2 sobre 2

La siguiente función se propone tomar como parámetro el nombre de un empleado y los importes de los últimos sueldos que percibió para luego mostrar un listado que incluya el nombre recibido y el promedio de sus sueldos:

```
def procesar(nombre, *importes):
    print('Nombre del empleado:', nombre)

    p = 0
    n = len(importes)
    if n != 0:
        s = 0
        for imp in importes:
            s += imp
        p = s / n
    print('Sueldo promedio:', p)
```

¿Cuáles de las siguientes invocaciones son correctas, en el sentido de ejecutarse sin problemas y mostrar los resultados pedidos sin provocar una interrupción del programa? (Observación: más de una respuesta puede ser válida, por lo que marque todas las que considere oportunas).

Seleccione una o más de una:

- a. `procesar('Laura', 1000, '2000', True)`
- b. `procesar('Carlos', '1000', '2000', '3000')`
- c. `procesar('Pedro', 1234.56, 2345.45)` ✓

  Correcto. La tupla ***importes*** tiene dos valores numéricos de tipo *float*, que son procesados correctamente, entregando un sueldo promedio de 1208.1866666666667

- d. `procesar('Luis')` ✓

  Correcto. En este caso, la tupla ***importes*** está vacía, y se muestra un sueldo promedio igual a cero, correctamente.

¡Correcto!

Las respuestas correctas son:
`procesar('Luis')`,

`procesar('Pedro', 1234.56, 2345.45)`

Pregunta 4

Correcta

Puntúa 1 sobre 1

¿Cuál de las siguientes afirmaciones es ***incorrecta*** en relación al concepto de *módulo* en Python, y/o a elementos asociados al uso de módulos en Python?

Seleccione una:

a.
Un módulo pensado para *ser ejecutado en forma directa* debería contener un script de control de la forma *if __name__ == '__main__'* para evitar que al ser importado se ejecute en forma inconveniente su posible script principal.

b.
Cuando se usa una instrucción import, Python busca primero si existe un módulo estándar cuyo nombre coincida con el del módulo importado. Si no lo encuentra, busca entonces en la lista de carpetas indicada por la variable sys.path.

c.
Un módulo en Python es cualquier archivo con extensión *.py* (código fuente que contenga funciones, definiciones generales, clases, variables, etc.)

d.
Un módulo siempre puede ser importado desde otro módulo, mediante instrucciones *import* o *from import*.

e.
Un módulo en Python puede tener elementos *docstring* que documenten su uso y su contenido.

f.
Cada vez que un módulo es importado en un programa, su contenido es vuelto a compilar por el intérprete. ✓

¡Correcto! Esta es justamente la afirmación incorrecta pedida: no es cierto que un módulo se compila nuevamente cada vez que se lo importa. Sólo se compila la primera vez, y esa precompilación se almacena en la carpeta __pycache__ del proyecto. Con esto se gana tiempo de ejecución, al no tener que compilar una y otra vez un módulo cada vez que un programa lo importa o lo accede.

¡Correcto!

La respuesta correcta es:
Cada vez que un módulo es importado en un programa, su contenido es vuelto a compilar por el intérprete.

Pregunta 5

Correcta

Puntúa 1 sobre 1

Suponga las siguientes instrucciones *import* en un programa (note que todos los módulos nombrados en esos *import* pertenecen a la librería estándar de Python):

```
import math
from re import split
from random import random
```

¿Cuáles de las siguientes invocaciones a funciones son ***correctas***, considerando los *import* que se acaban de mostrar? (Observación: más de una respuesta puede ser válida. Marque todas las que considere oportunas)

Seleccione una o más de una:

- [x] a.

```
# observación: la función split() (partir en subcadenas) pertenece al modulo re
y = split("\L+", "La ola de la vida")
```

✓

Correcto. Hay una instrucción de importación que específicamente introduce el nombre de la función split() en este script, por lo cual no debe usarse el prefijo "***re.***"

- [ ] b.

```
# observación: la funcion random() pertenece al modulo random
z = random.random()
```

- [x] c.

```
# obervacion: la funcion sqrt() (raiz cuadrada) pertenece al modulo math
x = math.sqrt(4)
```

✓

Correcto. Hay un import que introduce el nombre del módulo math en este programa, y a partir de allí cada función de ese módulo debe ser nombrada con el prefijo "***math.***"

- [ ] d.

```
# observacion: la funcion log2() (logaritmo en base 2) pertenece al modulo math
lg = log2(8)
```

¡Correcto!

Las respuestas correctas son:

```
# obervacion: la funcion sqrt() (raiz cuadrada) pertenece al modulo math
x = math.sqrt(4),
```

```
# observación: la función split() (partir en subcadenas) pertenece al modulo re
y = split("\L+", "La ola de la vida")
```

Pregunta 6

Correcta

Puntúa 1 sobre 1

En la columna de la izquierda, se nombran algunos de los módulos que existen en la librería estándar de Python. Seleccione el uso o aplicación de cada uno de estos módulos.

`sys`	Variables de entorno y acceso a parámetros de línea de órdenes
`doctest`	Validaciones de tests incluidos en strings de documentación.
`datetime`	Manipulación de fechas y horas
`urllib.request`	Recuperación de datos desde un url.
`re`	Reconocimiento de patrones
`xml.dom - xml.sax`	Parsing de documentos XML.
`os`	Acceso a funciones del sistema operativo.

¡Correcto!

La respuesta correcta es: `sys` → Variables de entorno y acceso a parámetros de línea de órdenes, `doctest` → Validaciones de tests incluidos en strings de documentación., `datetime` → Manipulación de fechas y horas, `urllib.request` → Recuperación de datos desde un url., `re` → Reconocimiento de patrones, `xml.dom - xml.sax` → Parsing de documentos XML., `os` → Acceso a funciones del sistema operativo.

Pregunta 7

Correcta

Puntúa 2 sobre 2

Suponga la definición del siguiente módulo en Python, ***tal como se muestra***, y suponga también que este módulo se ha almacenado en el archivo "*cadenas.py*":

```
def mensaje(m):
    print('El mensaje es:', m)

def invertir(m):
    print('El mensaje (invertido) es:')

    n = len(m)
    for i in range(n-1, -1, -1):
        print(m[i], end='')

def test():
    cadena = input('Ingrese un mensaje: ')
    mensaje(cadena)
    invertir(cadena)

test()
```

Suponga ahora que se define otro módulo (almacenado en el archivo "*prueba.py*" y en la misma carpeta que el módulo "*cadenas.py*") cuyo contenido es el siguiente, ***tal como se muestra***:

```
import cadenas

def principal():
    cad1 = 'Universidad Tecnologica Nacional'
    cadenas.mensaje(cad1)
    cadenas.invertir(cad1)

if __name__ == '__main__':
    principal()
```

¿Cuál de las siguientes afirmaciones describe correctamente lo que ocurrirá si se pide ejecutar el contenido del módulo "*prueba.py*"?

Seleccione una:

a.
El módulo prueba.py está importando al módulo cadenas.py y como este último ***no*** incluye un chequeo de la forma *if __name__ == "__main__"*, entonces al ejecutar prueba.py se ejecutará primero la función prueba.principal() y luego la función cadenas.test().

b.
El módulo prueba.py está importando al módulo cadenas.py y como este último ***no*** incluye un chequeo de la forma *if __name__ == "__main__"*, entonces al ejecutar prueba.py se producirá un error de intérprete cuando se intente cargar el módulo cadenas.py.

c.

El módulo prueba.py está importando al módulo cadenas.py y como prueba.py incluye un chequeo de la forma *if __name__ == '__main__'*, entonces al ejecutar prueba.py se ejecutará solamente la función prueba.principal().

d.

El módulo prueba.py está importando al módulo cadenas.py y como este último ***no*** incluye un chequeo de la forma *if __name__ == '__main__'*, entonces al ejecutar prueba.py se ejecutará primero la función cadenas.test() y luego la función prueba.principal(). ✓

¡Correcto!

¡Correcto!

La respuesta correcta es:
El módulo prueba.py está importando al módulo cadenas.py y como este último ***no*** incluye un chequeo de la forma *if __name__ == '__main__'*, entonces al ejecutar prueba.py se ejecutará primero la función cadenas.test() y luego la función prueba.principal().

Pregunta 8

Correcta

Puntúa 1 sobre 1

Para cada una de las variables predefinidas de uso global de Python de la primera columna, seleccione la definición que le corresponde o que mejor se le ajuste.

__author__	El nombre del autor del elemento. ✓
__name__	El nombre de un módulo o el valor literal '__main__'. ✓
__all__	Los elementos a importar desde un paquete. ✓
__doc__	Todos los docstrings que contiene el elemento. ✓

¡Correcto!

La respuesta correcta es: __author__ → El nombre del autor del elemento., __name__ → El nombre de un módulo o el valor literal '__main__'., __all__ → Los elementos a importar desde un paquete., __doc__ → Todos los docstrings que contiene el elemento.

Pregunta 9

Correcta

Puntúa 1 sobre 1

Suponga que se tiene un *paquete* en Python llamado ***soporte***, y que ese paquete contiene cuatro módulos llamados respectivamente *modelo*, *persistencia*, *interfaz* y *excepciones*. ¿Cuáles de las siguientes **son correctas** en relación al archivo ***__init__.py*** del paquete ***soporte***? (Más de una puede ser válida... marque todas las que considere apropiadas)

Seleccione una o más de una:

- a.
  El archivo __init__.py del paquete ***soporte*** no puede dejarse vacío. Debe asignar los nombres de los módulos que posee, en la variable ***__all__***.

- b.
  El archivo __init__.py del paquete ***soporte*** puede dejarse vacío. No importa si el paquete tiene subpaquetes y módulos o no. ✓

  Correcto... dejar vacío el archivo __init__.py es válido... aunque a veces podría no tener mucha utilidad.

- c.
  El archivo __init__.py del paquete ***soporte*** debe estar presente en la carpeta de ese paquete, para que Python lo reconozca como un paquete válido. ✓

  Correcto... justamente, la presencia de ese archivo es lo que hace que Python considere a la carpeta como un package.

- d.
  El archivo __init__.py del paquete ***soporte*** *puede* contener una asignación con los nombres de todos o algunos de los módulos que posee, en la variable ***__all__***. ✓

  Correcto... sabemos que se puede usar la variable ***__all__*** para asignar en ella los nombres de los subpaquetes y/o módulos que queremos que se importen si se hace un ***from soporte import ****.

¡Correcto!

Las respuestas correctas son:
El archivo __init__.py del paquete ***soporte*** puede dejarse vacío. No importa si el paquete tiene subpaquetes y módulos o no.,

El archivo __init__.py del paquete ***soporte*** *puede* contener una asignación con los nombres de todos o algunos de los módulos que posee, en la variable ***__all__***.,

El archivo __init__.py del paquete ***soporte*** debe estar presente en la carpeta de ese paquete, para que Python lo reconozca como un paquete válido.

Pregunta 10

Correcta

Puntúa 1 sobre 1

Suponga que se tiene un paquete llamado *test* en Python, y que ese paquete contiene tres subpaquetes llamados *prueba1*, *prueba2* y *prueba3*. Sunpoga además que archivo *__init_.py* del paquete *test* contiene solo el siguiente *docstring* (y ninguna otra asignación o script adicional):

```
"""El paquete contiene subpaquetes para operaciones de
testing de un sistema.
Lista de subpaquetes incluidos:
:prueba1: Contiene un modulo con funciones para testear
operaciones de input/output
:prueba2: Contiene dos modulos con funciones para
testear manejo de excepciones
"""
```

¿Hay algún problema con este bloque, en relación al respeto de las convenciones de documentación *docstring*? (Marque la respuesta que mejor describa la situación)

Seleccione una:

a.
Hay un par de problemas: debería haber una línea en blanco después de la primera línea del bloque (la descripción de los subpaquetes debería aparecer luego de esa línea en blanco); y además, faltaría la descripción del subpaquete *prueba3*. ✓

Correcto. Las convenciones generales sugieren la línea en blanco faltante, y las convenciones para describir un paquete sugieren que se documente brevemente todo subpaquete que el paquete contenga (salvo que algún subpaquete se haya excluido por no haberlo asignado en la variable *__all__*... ¡que no es el caso!)

b.
No hay ningún problema.

c.
Hay un solo y único problema: debería haber una línea en blanco después de la primera línea del bloque (la descripción de los subpaquetes debería aparecer luego de esa línea en blanco)

d.
El problema es el propio bloque en sí mismo: un paquete no lleva documentación *docstring*.

¡Correcto!

La respuesta correcta es:
Hay un par de problemas: debería haber una línea en blanco después de la primera línea del bloque (la descripción de los subpaquetes debería aparecer luego de esa línea en blanco); y además, faltaría la descripción del subpaquete *prueba3*.