Prueba Módulo 4

¡Bien hecho! Has llegado al final del Módulo 4 y también has terminado una parte importante de tu educación sobre programación en Python. A continuación se presenta un corto resumen de los objetivos cubiertos en el Módulo 4:

- estructuración del código y el concepto de funciones;
- invocación de funciones y devolución del resultado de una función;
- alcances de nombres y sombreado de variables;
- tuplas y su propósito, construcción y uso de tuplas;
- diccionarios y su finalidad, construcción y uso de diccionarios;
- excepciones - la sentencia *try* y la clausula *except* , excepciones integradas de Python, pruebas de código y depuración.

Ahora estás listo para tomar la **Prueba del Módulo**, que te ayudará a medir lo que has aprendido hasta ahora.

La siguiente prueba se basa en lo que acabas de aprender. Hay veintidós preguntas en total y debes obtener al menos un 70% para aprobar.

¡Buena suerte!

Pregunta 1

¿Cuál de las siguientes líneas inicia correctamente una definición de función sin parámetros?

- `def fun:`
- ✓ `def fun():`
- `function fun():`
- `fun function():`

Pregunta 2

Una función definida de la siguiente manera: (Elegir **dos** respuestas)

```
1  def function(x=0):
2      return x
3
```

- debe invocarse sin ningún argumento.
- debe ser invocada con exactamente un argumento.
- ✓ puede ser invocada sin ningún argumento.
- ✓ puede ser invocado con exactamente un argumento.

Pregunta 3

Una función integrada es una función que:

- ✓ viene con Python, y es una parte integral de Python
- tiene que ser importado antes de su uso
- está oculto a los programadores
- ha sido colocado dentro de tu código por otro programador

Pregunta 4

El hecho de que las tuplas pertenezcan a tipos de secuencia significa que:

- ✗ se pueden modificar usando la instrucción `del`
- en realidad son listas
- se pueden indexar y rebanar como las listas
- se pueden extender usando el método `.append()`

Pregunta 5

¿Cuál es la salida del siguiente fragmento de código?

```
1  def f(x):
2      if x == 0:
3          return 0
4      return x + f(x - 1)
5
6
7  print(f(3))
8
```

- `1`
- `3`
- el codigo es erroneo
- ✓ `6`

Pregunta 6

¿Cuál es la salida del siguiente fragmento de código?

```
def fun(x):
    x += 1
    return x


x = 2
x = fun(x + 1)
print(x)

```

- 5
- ✓ 4
- 3
- el código es erroneo

Pregunta 7

¿Qué código insertarías en lugar del comentario para obtener el resultado esperado?

Salida esperada:

```
a
b
c
```

Código:

```
dictionary = {}
my_list = ['a', 'b', 'c', 'd']

for i in range(len(my_list) - 1):
    dictionary[my_list[i]] = (my_list[i], )

for i in sorted(dictionary.keys()):
    k = dictionary[i]
    # Inserta tu código aquí.

```

- ✓ `print(k[0])`
- `print(k['0'])`

Pregunta 8

El siguiente fragmento de código:

```
def func(a, b):
    return a ** a


print(func(2))

```

- devolverá `None`
- dará como salida `2`
- ✓ es erroneo
- dará como salida `4`

El siguiente fragmento de código:

```
def func_1(a):
    return a ** a


def func_2(a):
    return func_1(a) * func_1(a)


print(func_2(2))

```

- dará como salida `4`
- es erroneo
- dará como salida `2`
- ✓ dará como salida `16`

Pregunta 10

¿Cuál de las siguientes líneas inicia correctamente una función utilizando dos parámetros, ambos con valores predeterminados de cero?

- ✓ `def fun(a=0, b=0):`
- `def fun(a=b=0):`
- `fun fun(a, b=0):`
- `fun fun(a=0, b):`

Pregunta 11

¿Cuáles de las siguientes afirmaciones son verdaderas? (Selecciona dos respuestas)

- El valor `None` puede ser empleado como argumento de operaciones aritméticas
- ✓ El valor `None` puede ser asignado a variables
- El valor `None` no puede ser empleado fuera de las funciones
- ✓ El valor `None` puede ser comparado con otras variables

Pregunta 12

¿Cuál es la salida del siguiente fragmento de código?

```
def fun(x):
    if x % 2 == 0:
        return 1
    else:
        return


print(fun(fun(2)) + 1)

```

- 1
- 2
- ✓ el código provocará un error de tiempo de ejecución
- None

Pregunta 13

¿Cuál es la salida del siguiente fragmento de código?

```
def fun(x):
    global y
    y = x * x
    return y


fun(2)
print(y)

```

- 2
- el código provocará un error de tiempo de ejecución
- ✓ 4
- Ninguno

Question 14

¿Cuál es la salida del siguiente fragmento de código?

```
def any():
    print(var + 1, end='')


var = 1
any()
print(var)

```

- ✓ 21
- 22
- 12
- 11

Pregunta 15

Asumiendo que `my_tuple` es una tupla creada correctamente, el hecho de que las tuplas son inmutables significa que la siguiente instrucción:

```
my_tuple[1] = my_tuple[1] + my_tuple[0]

```

- puede ser ilegal si la tupla contiene cadenas
- es completamente correcta
- ✓ es ilegal
- puede ser ejecutada solo si la tupla contiene al menos dos elementos

Pregunta 16

¿Cuál es la salida del siguiente fragmento de código?

```
my_list = ['Mary', 'had', 'a', 'little', 'lamb']


def my_list(my_list):
    del my_list[3]
    my_list[3] = 'ram'


print(my_list(my_list))

```

- ✗ `['Mary', 'had', 'a', 'ram']`
- `['Mary', 'had', 'a', 'little', 'lamb']`
- no hay salida, el fragmento es erroneo

Pregunta 17

¿Cuál es la salida del siguiente fragmento de código?

```
def fun(x, y, z):
    return x + 2 * y + 3 * z


print(fun(0, z=1, y=3))

```

- `0`
- `3`
- ✓ `9`
- el código es erroneo

Pregunta 18

¿Cuál es la salida del siguiente fragmento de código?

```
def fun(inp=2, out=3):
    return inp * out


print(fun(out=2))

```

- [x] `4`
- [ ] `6`
- [ ] `2`
- [ ] el código es erroneo

Pregunta 19

¿Cuál es la salida del siguiente código?

```
dictionary = {'one': 'two', 'three': 'one', 'two': 'three'}
v = dictionary['one']

for k in range(len(dictionary)):
    v = dictionary[v]

print(v)

```

- [x] `two`
- [ ] `one`
- [ ] `('one', 'two', 'three')`
- [ ] `three`

Pregunta 20

¿Cuál es la salida del siguiente código?

```
tup = (1, 2, 4, 8)
tup = tup[1:-1]
tup = tup[0]
print(tup)

```

- [ ] `(2, )`
- [x] `2`
- [ ] `(2)`
- [ ] el código es erroneo

Pregunta 21

Selecciona las sentencia **true** sobre el bloque *try-except* en relación con el siguiente ejemplo. (Selecciona **dos** respuestas).

```
try:
    # Algo de código...
except:
    # Algo de código...

```

El código que sigue a la sentencia `try` será ejecutado si el código dentro de la sentencia `except` se encuentra con un error.

✓ Si sospechas que un fragmento de código puede generar una excepción, se debe colocar dentro del bloque `try`.

✓ El código que sigue a la sentencia `except` será ejecutado si el código en el bloque `try` se encuentra con un error.

Si existe un error de sintaxis en el código ubicado en el bloque `try`, la sentencia `except` **no** lo manejará, y una excepción *SyntaxError* será generada.

Pregunta 22

¿Cuál es la salida del siguiente código?

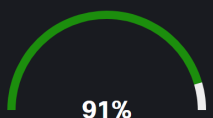
```
try:
    value = input("Ingresa un valor: ")
    print(value/value)
except ValueError:
    print("Entrada incorrecta...")
except ZeroDivisionError:
    print("Entrada errónea...")
except TypeError:
    print("Entrada muy errónea...")
except:
    print("¡Buuu!")
```

`Entrada incorrecta...`

✓ `Entrada muy errónea...`

`Entrada errónea...`

Revisar Evaluación



Has obtenido 91%.

Felicidades, has pasado la prueba.