1. Para la tarea de determinar si un usuario es mayor de edad o no, se dispone de la función *mayor(edad)* que recibe como argumento un número entero que representa la edad, y retorna True si la persona tiene 18 años o más, y False en caso contrario.
Determina cuál o cuáles de las siguientes alternativas presenta funciones que realicen lo anterior correctamente.

0 / 1 punto

☐
```
1  def mayor(edad):
2      print(edad >= 18)
```

☐
```
1  def mayor(edad):
2      return edad >= 18
```

☐
```
1  def mayor(edad):
2      if edad >= 18:
3          print(True)
4      else:
5          print(False)
```

☑
```
1  def mayor(edad):
2      if edad >= 18:
3          return True
4      else:
5          return False
```

Correcto
Se puede utilizar control de flujo para verificar la condición pedida y luego retornar explícitamente el valor booleano.

☐
```
1  def mayor(edad):
2      if edad >= 18:
3          return True
4      return False
```

No seleccionaste todas las respuestas correctas

2. Dada la función *factorial(n),* que dado un número natural *n,* retorna el valor de n!, es decir, 1*2*3*4*...*n, se desea que en la variable *resultado* esté almacenado el valor de dicho cálculo. Determina cuál de las siguientes alternativas es

1 / 1 punto

correcta para calcular el factorial de 5 y que quede en dicha variable.

- ```
  n = 5
  resultado = 0
  factorial(n, resultado)
  ```

- ```
  factorial(5)
  ```

- ```
  resultado = 5
  factorial(resultado)
  ```

- ```
  resultado = 5
  n = factorial(resultado)
  ```

- ```
  n = 5
  resultado = factorial(n)
  ```

Correcto
La variable resultado está igualada a la función, y de este modo adquiere el valor de retorno de la función.

3. En cuanto al *scope* de funciones. Determina cuáles de los siguientes códigos termina en un error de Python. 1 / 1 punto
  ```

- [ ]
```
def sumador(n, sumando):
    sumando += 1
    n += sumando
    return n
b = 9
sumador(5, b)
```

- [ ]
```
sumando = 10
def sumador(n):
    n += sumando
    return n
sumador(5)
print(sumando)
```

- [x]
```
cantidad = 0
def sumador(n):
    cantidad += 1
    n += 1
    return n
sumador(5)
```

Correcto
En el llamado a *sumador* la función intenta modificar la variable cantidad, pero esta vive fuera del *scope* de dicha función, de modo que no se puede modificar de esa manera.

- [ ]
```
def sumador(n):
    sumando = 10
    n += sumando
    return n
sumador(5)
```

- [x]
```
def sumador(n):
    sumando = 10
    n += sumando
    return n
sumador(5)
print(sumando)
```

Correcto
Al intentar imprimir la variable *sumando* en la línea 6, el programa arrojará error ya que la variable *sumando* solo fue definida dentro de la función sumador, es decir, solo vive en ese *scope,* y al terminar la función, se "muere" dicha variable.

4. Determina lo que imprimirá el siguiente código: 1 / 1 punto

```
numero = 10
def operacion(n):
    numero = 100
```

```
    numero = 100
    return n * numero
operacion(5)
print(numero)
```

- (●) 10
- ( ) 100
- ( ) Depende del valor de n
- ( ) Arrojará error de Python

Correcto
Se imprime el valor de la variable fuera de la función, ya que lo que sucede dentro de la misma, solo vive ahí, de modo que en el mundo exterior de dicha función, la variable no ha sido modificada.

5. Si se tiene un módulo de funciones de nombre *modulo.py,* y este contiene las funciones *multiplicar(a, b)* y *dividir(a, b).* Determina cuáles de los siguientes códigos es correcto. 0 / 1 punto

- [x]
```
from modulo import *
print(modulo.multiplicar(2, 3))
print(modulo.dividir(10, 5))
```

Esto no debería estar seleccionado
Es incorrecto ya que al importar el módulo con * se importan con el nombre directo de las funciones y no es necesario indicar el módulo del que pertenecen como prefijo.

- [ ]
```
import modulo
print(multiplicar(2, 3))
print(dividir(10, 5))
```

- [x]
```
from modulo import *
print(multiplicar(2, 3))
print(dividir(10, 5))
```

Correcto
Es correcto ya que al indicar * se importan todas las funciones del módulo con el nombre con que están definidas en el mismo.

- [x]
```
from modulo import multiplicar, dividir
print(multiplicar(2, 3))
print(dividir(10, 5))
```

Correcto
Es correcto ya que se importan las funciones específicas utilizando el nombre, y separadas por coma.

- [ ]
```
import modulo
print(modulo.multiplicar(2, 3))
print(modulo.dividir(10, 5))
```

6. Determina el valor que queda almacenado en las variables *resultado1* y *resultado2* tras la ejecución del siguiente código: 1 / 1 punto

```
def operacion(n):
  if n > 10:
    return 20
    return 15
  return 10
  return 25

resultado1 = operacion(8)
resultado2 = operacion(12)
```

- resultado1 = 8
  resultado2 = 12
- resultado1 = 25
  resultado2 = 20
- resultado1 = 25
  resultado2 = 15
- [x] resultado1 = 10
  resultado2 = 20
- resultado1 = 10
  resultado2 = 15

Correcto
Las funciones al llegar al una sentencia *return* terminan su ejecución, por lo que se quedarán siempre con el primer valor de retorno que encuentre.

7. Considera el siguiente código: 1 / 1 punto

```
def funcion(x):
  n = 3
```

```
3     return ?
4   print(funcion(9))
5   print(funcion(10))
```

¿Qué debe retornar la función en lugar de ese "?" para que el código imprima True y False respectivamente?

- ○ x == n
- ○ bool(x % n)
- ○ x % n
- ◉ not bool(x % n)
- ○ x != n

✓ Correcto
Es correcto, así la función se preocuparía de verificar si un número *x* es divisible por 3 o no, viendo si el resto de la división es 0, cuyo valor booleano es False, o mayor que cero, cuyo valor booleano es True.

8. Considera el siguiente código: 1 / 1 punto

```
1   numero = 5
2   resultado = exponenciacion_aleatoria(numero)
3
4   def exponenciacion_aleatoria(n):
5     return n ** random.randint(1, 10)
```

Selecciona todas las alternativas que muestren razones por las cuales el código anterior es incorrecto.

- ☐ Se utiliza un "_" en el nombre de la función
- ☐ Se utiliza un operador inválido: **
- ☑ No se importa el módulo random

✓ Correcto
Los módulos deben ser importados antes de ser utilizados.

- ☐ Se realiza una operación en el retorno, en vez de realizarla antes y retornar una variable que almacene el resultado
- ☑ Se llama a la función antes de que se hay definido

✓ Correcto
Las funciones deben ser definidas antes de que sean llamadas.

9. Selecciona la afirmación incorrecta respecto a funciones. 1 / 1 punto

- ○ Las variables definidas dentro de una función no pueden ser utilizadas fuera de ella.
- ○ Las variables definidas fuera de una función no pueden ser modificadas dentro de una función.
- ○ Los llamados a funciones deben hacerse después de la definición de la función.
- ◉ Una función no puede tener más de dos retornos.

El retorno de una función puede incluir expresiones booleanas.

Correcto
Puede tener una cantidad arbitraria de retornos.

10. Considerando el siguiente programa: 1 / 1 punto

```
1  def funcion(n):
2      a = n ** 3
3      b = a ** 2
4      c = b + 100
5      d = 5 * c
6      return print(d)
7
8  d = funcion(2)
```

Determina el valor que queda almacenado en *d* tras la ejecución del programa.

- 820
- None
- 0
- 2
- Se genera un error de Python

Correcto
print es una función que no retorna un valor, sino que retorna None. Por lo tanto, al retornar una ejecución de la función *print*, lo que va a suceder es que en la variable *d* queda almacenado el valor de retorno de *print*, es decir, None.