

Midterm

Compilado preguntas de alternativas

A continuación se presenta un listado de preguntas de alternativas, que les permita conocer el tipo de preguntas que se realizarán durante el *Midterm* y además prepararse para esta evaluación que se realizará el **17 de octubre**.

De forma excepcional, algunas preguntas de este compilado presentan múltiples respuestas correctas, por lo que en dichos casos deberán selección más de una alternativa (si corresponde). Por el contrario, durante el *Midterm* todas las preguntas tendrán una **única alternativa correcta**.

Índice

2. Respuestas

,	guntas
	Programación Orientada a Objetos I
1.2.	Programación Orientada a Objetos II
1.3.	Built-ins, Iterables y Funcional
1.4.	Interfaces Gráficas I
1.5.	Threading
1.6.	Interfaces Gráficas II
1.7.	Serialización y Excepciones

13

1. Preguntas

1.1. Programación Orientada a Objetos I

- 1. ¿Cuál(es) de las siguientes afirmaciones es/son falsa(s) en torno al concepto de properties?
 - (a) La utilización de getters y setters violan el principio de encapsulamiento y por ende hay que tener cuidado al implementarlos.
 - B Las properties tienen la interfaz de un atributo.
 - C) Toda property debe tener un getter. 🗸
 - En Python, solo se pueden definir properties a través del decorador @property.
- 2. ¿Cuál(es) es/son el/los error(es) en el siguiente código?

- A) self.carrera no está definido en el inicializador.
- B) edad no está definido.
- C) Falta implementar herencia desde alguna otra clase llamada Joven.
- D) El código seguirá corriendo de forma indefinida.

3. A partir del código que se presenta a continuación:

```
class Variable:
1
        def __init__(self, valor):
2
            self._valor = valor
3
4
        @property
5
        def valor(self):
6
            return self._valor
        @valor.setter
9
        def valor(self, x):
10
            if x <= 0:
11
                 self.\_valor = x
12
            else:
13
                 self.\_valor = -x
14
15
    variable_uno = Variable(5) 5
16
    # Paso 1
17
    variable_uno.valor -= 3 - 1
18
    # Paso 2
19
    variable_uno.valor = -1 - 1
20
    # Paso 3
^{21}
    variable_uno.valor += 3 🛂
```

¿Cuál es el valor del atributo valor después de que se ejecuten los 3 pasos?

- \bigcirc -2
- B) 2
- C) 1
- D) -1

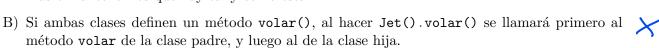
4. ¿Qué se imprimirá en consola al ejecutar este código?

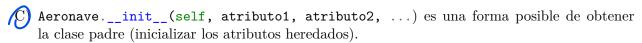
```
class Auto:
1
         def __init__(self):
2
             self.marca = ""
3
             self.motor = ""
4
             self.color = ""
5
6
         def __repr__(self):
             return(f"Mi marca es {self.marca}")
9
    spark = Auto()
10
    spark.marca = "chevrolet"
11
    spark.patente = "GT HB 34" St Se puece print(spark patents)
12
    print(spark.patente)
13
```

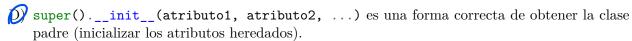
- A) GT HB 34
- B) AttributeError: 'Auto' object has no attribute 'patente'
- C) Mi marca es chevrolet
- D) chevrolet

1.2. Programación Orientada a Objetos II

- 5. Si creamos una subclase Jet que hereda de una clase Aeronave, ¿cuál(es) de las siguientes afirmaciones es/son verdadera(s) respecto a la subclase?
 - A) No se pueden modificar los métodos de la clase padre en la subclase, solo se pueden agregar más o mantener los que hay tal y como están.







- 6. ¿Cuándo se produce el problema del diamante y cómo se soluciona?
 - A) Se produce al utilizar clases abstractas y se soluciona llamando al super().metodo() en cualquier método que lo necesitemos.
 - B) Se produce al utilizar multi-herencia y se soluciona llamando al super().metodo() en cualquier método cuando lo necesitemos.
 - C) Se produce al utilizar clases abstractas y se soluciona llamando al super().__init__() solo en el __init__ que estamos definiendo.
 - D) Se produce al utilizar multi-herencia y se soluciona llamando al super().__init__() solo en el __init__ que estamos definiendo.

7. Se quiere modelar una granja que tiene diferentes tipos de animales. Para esto se crea la clase abstracta Animal y luego se crean tres subclases Chancho, Vaca y Pato, que heredan de Animal. Animal a su vez tiene un método abstracto llamado hablar() que las tres subclases deben sobrescribir, para que al llamarlo, el chancho diga "Oink!", la vaca diga "Muu!" y el pato diga "Quack!".

¿Cuál(es) de los siguientes conceptos **NO** se usa(n) para esta modelación?

- A) Encapsulamiento
- B) Herencia 🗸
- C) Polimorfismo ...
- (D))Properties
- 8. ¿Para qué **NO** sirven las clases abstractas?
 - A) Para reducir código. 🗸
 - B) Para modelar las subclases.
 - (C) Para tener una clase que se pueda instanciar siempre.
 - D) Para evitar redundancia.

1.3. Built-ins, Iterables y Funcional

9. ¿Cuáles de los siguientes códigos son declaraciones válidas de funciones en Python? (Es decir, que no tienen errores de sintaxis en Python)

```
def funcion(*args, primero=None, **kwargs):
    pass
```

- B) def funcion(*args, **kwargs, primero): ×
 pass
- C) def funcion(*args, **kwargs, primero=None): X
 pass
- D) def funcion(*args, primero, **kwargs): X pass
- def funcion(primero, *args, **kwargs):

 pass

10. A partir del siguiente código:

```
iterable = [1, 2, 3, 4, 5]
    iter_a = iter(iterable)
2
    iter_b = iter(iterable)
    lista = []
4
5
    for i in iter_a:
6
        lista.append(i)
        if i >= 3:
             break
9
10
    for j in iter_b:
11
        lista.append(j)
12
        if j >= 2:
13
             break
14
15
    for k in iter_a:
16
        lista.append(k)
17
        if k \ge 4:
18
             break
19
20
    print(lista)
^{21}
```

El *output* esperado es:

- A) [1, 2, 3, 1, 2]
- B) [1, 2, 3, 1, 2, 1, 2, 3, 4]
- (i) [1, 2, 3, 1, 2, 4]
- D) [1, 2, 3, 4]

11. ¿Cuál es el *output* del siguiente código?

```
nombres = ["DCCachorritos", "DIElefante", "DCCorales", "ICMagia"]
map_object = map(lambda s: s[0:3] == "DCC", nombres)
print(list(zip((map_object))))
```

- (True,), (False,), (True,), (False,)]
- B) [True, False, True, False]
- C) (True, False, True, False)
- D) [(True), (False), (True), (False)]

12.	${\it Marque todas las alternativas correctas sobre los parámetros * \tt args y **kwargs en una función.}$			
	A	*args permite recibir argumentos posicionales y **kwargs permite recibir argumentos por palabra clave.		
	B)	*args permite recibir argumentos por palabra clave y **kwargs permite recibir argumentos posicionales.		
	C)	Cuando están ambos presentes, la función se debe llamar con, al menos un argumento posicional, y al menos un argumento por palabra clave.		
	D	Si una función desea recibir un cantidad indeterminada de parámetros, puede usar solamente *args, o solamente **kwargs, o ambos.		
	E)	args y kwargs son palabras reservadas de Python. 🗙		
13.	¿Си	ál es la diferencia entre un iterador y un iterable ?		
		Un iterable es cualquier objeto que se puede iterar, mientras que un iterador es el objeto que itera sobre el iterable.		
	B)	Un iterador es cualquier objeto que se puede iterar, mientras que un iterable es el objeto que itera sobre el iterador.		
	C)	Ambos refieren al mismo objeto, solo que iterable se le llama en versiones antiguas de Python mientras que iterador es el término correcto según PEP8.		
	D)	Un iterable es una estructura ya implementada en Python, mientras que iterador es una estructura que se puede implementar con el uso de clases.		
14.	_	uieres obtener el resultado de una función sobre cada uno de los elementos en un iterable, la ción que mejor se ajusta a tu objetivo es:		
	(A)	reduce		
	B)	map		
	C)	filter		
	D)	lambda		
15.	¿Си	ál de las siguientes afirmaciones respecto a yield es correcta?		
	A)	La sentencia yield es igual a hacer un print. 🗶		
	B)	La sentencia yield es exactamente igual a un return. 🗶		

C) yield "resetea" los valores cada vez que llamo a la función. $\boldsymbol{\varkappa}$

Dyield empieza desde el valor anterior cada vez que llamo a la función.

Interfaces Gráficas I

16. Si dentro de una Ventana (QWidget), hay un método para abrir otra ventana y esconder la actual, ¿por qué este no funciona correctamente?

```
def abrir_otra_ventana(self):
       self.hide() # Esconder la ventana actual
2
       otra_ventana = Ventana("Otra ventana", 300, 100) # Crear otra
3
       otra ventana.show() # Mostrar nueva ventana
```

- A) hide() y show() deben ir cambiados en posición.
- B) La instancia de Ventana no está definida correctamente.
- C) otra_ventana es una variable local del método y se descarta.
- D) show() no es la manera correcta de mostrar una ventana. X
- 17. ¿A través de cuál método se puede conocer el objeto que envió una señal?
 - (A) sender()
 - B) signal.text()
 - C) signal()
 - D) sender.text()
- 18. ¿Cuál de las siguientes señales permite enviar un diccionario, un string y una tupla?
 - A) senal = pyqtSignal(self.dicc, self.string, self.tupla) donde self.dicc, self.str, self.tupla corresponden a atributos del tipo dict, str y tuple respectivamente.
 - B) senal = pyqtSignal()
 - senal = pyqtSignal(dict, str, tuple)
 - D) senal = pyqtSignal(*args)
 - E) senal = signal(dict, str, tuple)
- 19. ¿Cuál o cuáles de los siguientes elementos responden a acciones del usuario en la GUI?
 - A) QHBoxLayout X
 - B) QPushButton 🗸
 - C) QLabel

 - D) QLineEdit
 E) QObject X My amplo?

1.5. Threading

20. De acuerdo al código siguiente, marque todas las alternativas correctas:

```
lock_comida = threading.Lock()
2
    def comer(nombre, comida, lock):
3
        print(f"{nombre} está esperando para comer...")
4
        time.sleep(1)
5
        lock.acquire()
6
        print(f"{nombre} está comiendo {comida}")
        time.sleep(3)
        print(f"{nombre} terminó de comer")
9
10
    pepito = threading.Thread(target=comer, args=("Pepito", "papas", lock_comida,))
11
    juan = threading.Thread(target=comer, args=("Juan", "pizza", lock_comida,))
12
    pepito.start()
13
    juan.start()
14
    pepito.join()
15
    juan.join()
    print("Ambos comieron")
```

- Se instancian correctamente los threads pepito y juan.
- B) Si se corre el código se imprimirá "Ambos comieron".
- Si se agrega lock.release() después de la línea 9 (dentro de la función comer), se imprimirá "Ambos comieron".
- D) Si se agrega lock.release() después de la línea 9 (dentro de la función comer), Juan siempre va a "comer" antes que Pepito.
- 21. ¿Cuál(es) de las siguientes afirmaciones es/son falsa(s)?
 - Dos threads distintos no pueden compartir un mismo Lock. X
 - B Se puede ejecutar un mismo thread más de una vez.
 - El método join() solo puede ser llamado por el main thread.
 - D Los daemon threads impiden que el programa termine si aún están corriendo.

22. ¿Qué ocurre una vez que se instancia un thread usando esta instrucción?

```
T = threading.Thread(target = func)
```

- Ahora T es una instancia de Thread, pero solo ejecutará func cuando se llame al método start().
- B) El thread principal espera hasta que termine la ejecución de la función de func. X
- C) El thread principal y la función func continúan ejecutándose simultáneamente.
- D) Si hay más de 1 núcleo disponible, entonces el *thread* principal y la función **func** se ejecutan en paralelo. De lo contrario, uno espera hasta que el otro se ejecute completamente.
- 23. Marque todas las opciones correctas para que un thread adquiera un lock llamado mi_lock.

```
A mi_lock.acquire()
B) mi_lock.release() 
O with mi_lock:
```

D) mi_lock.join() 💉

24. Te piden crear un juego donde un personaje debe realizar una misión. El juego termina cuando la misión sea completada o a los 100 segundos de juego, lo que pase primero. El tiempo es contado en un *thread* reloj aparte, cuyo *target* es la siguiente función:

```
def contar():
    tiempo = 0
    while tiempo < 100:
        time.sleep(1)
        tiempo += 1
    return "TIEMPO ACABADO"</pre>
```

¿Cómo deberías crear el thread para que no ocasione problemas al terminar el juego?

```
A) reloj = threading.Thread(target=contar)
B) reloj = threading.Thread(target=contar, daemon=False)
C reloj = threading.Thread(target=contar, daemon=True)
D) reloj = threading.Thread(target=daemon)
```

1.6. Interfaces Gráficas II

- 25. ¿Cuál de las siguientes declaraciones es correcta respecto a Threads versus QThreads?
 - A) Los métodos de QThreads son exactamente los mismos que tiene Threads. 🔀
 - B) Usar un Thread en conjunto a PyQt6 es lo mismo que usar un QThread, por lo que no importa cuál se use.
 - C) La clase QThread es subclase de Thread.
 - D El método is_alive no existe en QThread, en cambio, sí existe en Thread.

is running

- 26. ¿Cuál de las siguientes declaraciones es incorrecta respecto a una QMainWindow?
 - A) Para agregar el contenido principal de una QMainWindow se utiliza setCentralWidget(QWidget), la cual soporta cualquier QWidget.
 - B) Las QActions son comandos que pueden ser invocados por barras de menú y de herramientas, entre otros.
 - C Se muestra un mensaje en una barra de estado utilizando showText(str). Show message?
 - D) Se agrega una QAction a un componente de la QMainWindow utilizando addAction(QAction).
- 27. ¿Cuál de estos eventos está correctamente conectado?
 - A) mi_qthread = QThread(...)
 mi_qthread.timeout.connect(funcion)
 - B) mi_senal = pyqtSignal(...)
 mi_senal.clicked.connect(funcion)
 - mi_qtimer = QTimer(...)
 mi_qtimer.timeout.connect(funcion)
 - D) mi_boton = QPushButton(...)
 mi_boton.connect(funcion)
- 28. ¿Cuál es la principal diferencia entre QTimer de PyQt y Timer de threading?
 - A) QTimer se ejecuta con el método start(), en cambio Timer se ejecuta con el método run().
 - B) Timer ejecuta una subrutina una sola vez luego de una cierta cantidad de tiempo, en cambio QTimer ejecuta una subrutina periódicamente cada cierto tiempo.
 - C) Timer se ejecuta con el método start(), en cambio QTimer se ejecuta con el método begin(). x D
 - D) QTimer ejecuta una subrutina una sola vez luego de una cierta cantidad de tiempo, en cambio Timer ejecuta una subrutina periódicamente cada cierto tiempo.

1.7. Serialización y Excepciones

29. Considere el siguiente código:

```
def favoritometro(series, nombre):
    if not isinstance(nombre, str):
        raise KeyError as "Nombre erróneo"
    else:
        print(f"Serie favorita: {series[nombre]}")

dict_series = {"Alice": "Breaking Bad", "Bob": "Dark"}
    user = "uwu"
    favoritometro(dict_series, user)
```

¿Qué error aparece al correr el código anterior?

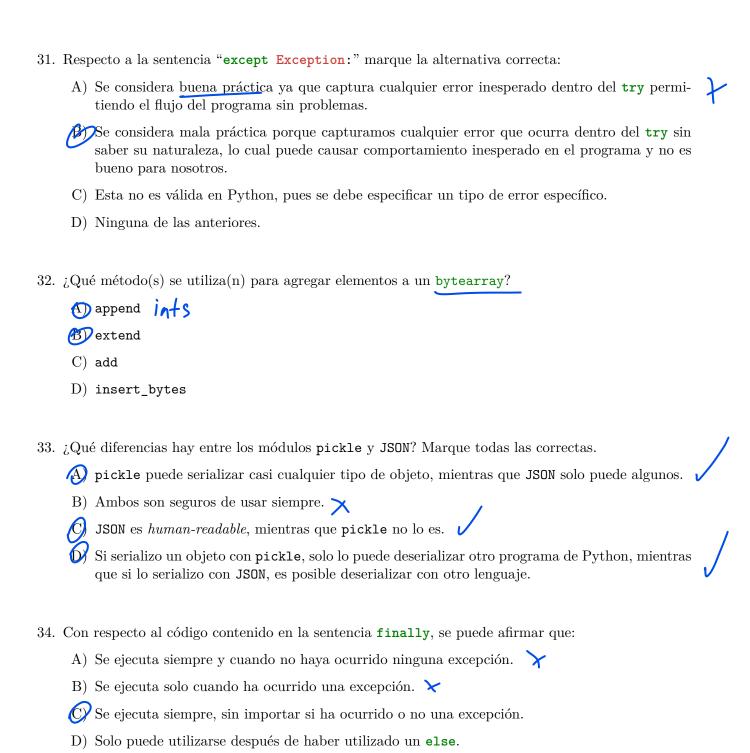
- (A) KeyError
- B) TypeError
- C) SyntaxError
- D) ValueError
- 30. Considere el siguiente código:

```
def calculadora(a, b):
    try:
        return a + b
        finally:
        return a * b

k = calculadora(2, 4)
print(k)
```

¿Qué se imprime?

- A) 2
- B) 6
- (C))8
- D) Nada



2. Respuestas

1. A y D

2. B

3. A

4. A

5. C y D

6. B

7. A y D

8. C

9. A, D y E

10. C

11. A

12. A y D

13. A

14. B

15. D

16. C

17. A

18. C

19. B, C y D

20. A y C

21. A, B, C y D

22. A

23. A y C

24. C

25. D

26. C

27. C

28. B

29. C

30. C

31. B

32. A y B

33. A, C y D

34. C