Universidad de los Andes
Facultad de Ingeniería

Departamento de Ingeniería de Sistemas y Computación
ISIS - 1222 – Introducción a la Programación Honores

ABET Engineering Accreditation Commission

N2-EXAM – 2025-20 – SOLUCIÓN

Pregunta 1 (30 pts). Implemente en Python la función `fecha_mas_reciente`, tal y como se describe en la documentación (siguiente página). No es necesario crear aserciones.

Lista de chequeo:

- ☐ La función incluye **2** doctests significativos y no redundantes.
- ☐ La función **no** usa funciones como `min()` y `max()`.
- ☐ La función usa el operador ternario y **no** usa condicionales.
- ☐ Se usa la comparación lexicográfica explicada a continuación.

Puede asumir que las tres fechas dadas siempre están en el formato `'YYYY-MM-DD'` (Año-Mes-Día). En Python, este formato permite que las fechas puedan compararse directamente como strings, ya que el orden lexicográfico coincide con el orden cronológico. Ejemplos de comparaciones:

```
"2005-02-15" < "2006-06-10"  # → True (Porque 2005 es anterior a 2006)
"2010-08-23" > "2009-12-31"  # → True (Porque 2010 es posterior a 2009)
"2015-03-10" < "2015-03-20"  # → True (Mismo año y mes, pero el día 10 es anterior al día 20)
```

La siguiente, es la rúbrica de calificación que se usará para calificar la pregunta 1:

Descripción	Cumple	Cumple parcialmente	Incumple
La implementación de la función cumple con lo solicitado: 30 pts si cumple a cabalidad ya que el algoritmo es correcto, cumple con todas las buenas prácticas del curso (BP-1 a BP-23) y con toda la lista de chequeo. 15 pts si el algoritmo es parcialmente correcto y falla en detalles muy menores y/o se incumple hasta una sola buena práctica del curso (BP-1 a BP-23) o un solo ítem de la lista de chequeo. 0 pts si el algoritmo tiene errores graves y/o se incumple más de una buena práctica del curso (BP-1 a BP-23) o más de un ítem de la lista de chequeo.			
Observaciones del profesor:			

```python
def fecha_mas_reciente(fecha1: str, fecha2: str, fecha3: str) -> str:
    """
    Retorna la fecha más reciente. En caso de empate, retorna cualquiera más reciente.
    Args:
        fecha1 (str): Primera fecha a comparar. Está en el formato 'YYYY-MM-DD'.
        fecha2 (str): Segunda fecha a comparar. Está en el formato 'YYYY-MM-DD'.
        fecha3 (str): Tercera fecha a comparar. Está en el formato 'YYYY-MM-DD'.
    Returns:
        str: La fecha más reciente (cronológicamente mayor) entre las tres.
```

Ejemplo de solución:

```python
    >>> fecha_mas_reciente("2021-01-01", "2022-01-01", "2023-01-01")  # Orden por año
    '2023-01-01'

    >>> fecha_mas_reciente("2015-05-20", "2015-12-01", "2015-11-30")  # Orden por mes (mismo año)
    '2015-12-01'

    >>> fecha_mas_reciente("2010-06-15", "2010-06-30", "2010-06-02")  # Orden por día (mismo año y mes)
    '2010-06-30'

    >>> fecha_mas_reciente("2019-03-10", "2019-03-10", "2018-12-31")  # Empate entre 2 fechas máximas
    '2019-03-10'

    >>> fecha_mas_reciente("2000-01-01", "2000-01-01", "2000-01-01")   # Empate entre 3 fechas máximas
    '2000-01-01'
    """
    return fecha1 if fecha2 <= fecha1 >= fecha3 else fecha2 if fecha1 <= fecha2 >= fecha3 else fecha3
```

Errores comunes:

- Violar la buena práctica BP-18. Tema que se recomendó explícitamente estudiar.
- Usar comillas dobles para establecer el resultado esperado del doctest.
- No usar comillas para establecer el resultado esperado del doctest.
- Nunca llamar a la función **`fecha_mas_reciente`** en los doctests.
- Hacer asignaciones dentro del operador ternario, sin entender que el operador ternario es una expresión y por lo tanto, resuelve a un valor.
- Usar el operador ternario con una sintaxis inválida (Ej: sin `else` al final). El operador ternario, es un tema que se recomendó explícitamente estudiar y del cual además se proyectó diapositivas de ejemplo durante el parcial.
- Usar valores en lugar de variables (Ej: `"fecha1"` en lugar de la variable, `fecha1`) . Esto indica una clara incomprensión del curso.
- Usar un algoritmo que no selecciona la fecha más reciente, sino cualquiera de las tres.
- No retornar nada.

Las preguntas 2 y 3 se basan en el proyecto de Nivel 2: **CupiTikTok**.

Utilice la siguiente información de contexto:

- El diccionario de un creador de contenido, tiene las siguientes llaves:

`nombre (str)`: Nombre del creador de contenido.
`pais (str)`: País de origen del creador de contenido.
`categorias (str)`: Categoría de los videos del creador de contenido.
`likes (int)`: Cantidad de likes que ha recibido el creador de contenido. Entero mayor o igual a 0.
`vistas (int)`: Cantidad de vistas que ha recibido el creador de contenido. Entero mayor o igual a 0.
`seguidores (int)`: Cantidad de seguidores que tiene el creador de contenido. Entero mayor o igual a 0.
`fecha_ultima_publicacion (str)`: Fecha de la última publicación del creador de contenido en formato `"YYYY-MM-DD"`. Ejemplo: `"2025-06-21"`. La fecha siempre está en ese formato.

- La siguiente es una implementación válida de la función `calcular_rating_creador()`:

```python
def calcular_rating_creador(creador: dict) -> float:
    """
    Calcula el rating de un creador de contenido con base en su número de seguidores, likes y vistas.
    Args:
        creador (dict): Diccionario que representa a un creador de contenido.
    Returns:
        float: El rating del creador de contenido, redondeado a dos cifras decimales.
            Este valor se encuentra entre 0 y 100.
    """
    S_MAX = 600_000
    L_MAX = 100_000_000
    V_MAX = 100_000_000
    rating = (
        (creador["seguidores"] / S_MAX) * 0.5
        + (creador["likes"] / L_MAX) * 0.3
        + (creador["vistas"] / V_MAX) * 0.2
    )

    return round(rating, 2)
```

Pregunta 2 (30 pts). Implemente en Python la función `filtrar_creadores_colombianos_destacados`, tal y como se describe en la documentación (siguiente página). Note que solo se usan dos creadores de contenido, en lugar de cuatro, para facilitarle la escritura de la función y que esta no sea muy extensa. **No** debe crear doctests.

Lista de chequeo:

- ☐ La función incluye aserciones que verifican el tipo de dato esperado de cada parámetro.
- ☐ La función incluye otras dos aserciones que verifican las condiciones sobre `anio`.
- ☐ La función **no usa** funciones como `min()` y `max()`.
- ☐ La función usa condicionales y **no** usa el operador ternario.

La siguiente, es la rúbrica de calificación que se usará para calificar la pregunta 2:

Descripción	Excelente	Bien	Regular	Insuficiente
La implementación de la función cumple con lo solicitado: 30 pts si cumple a cabalidad ya que el algoritmo es correcto, cumple con todas las buenas prácticas del curso (BP-1 a BP-23) y con toda la lista de chequeo. 20 pts si el algoritmo es parcialmente correcto y falla en detalles muy menores y/o se incumple hasta una sola buena práctica del curso (BP-1 a BP-23) o un solo ítem de la lista de chequeo. 10 pts si el algoritmo es parcialmente correcto con errores menores y/o se incumple hasta dos buenas prácticas del curso (BP-1 a BP-23) o dos ítems de la lista de chequeo. 0 pts si el algoritmo tiene errores graves y/o se incumple más de dos buenas prácticas del curso (BP-1 a BP-23) o más de dos ítems de la lista de chequeo.				
Observaciones del profesor:				

```
def filtrar_creadores_colombianos_destacados(anio: str, c1: dict, c2: dict) -> str:
    """
    Busca a los creadores de contenido COLOMBIANOS que cumplan con todas las siguientes condiciones:
        1. Tienen más de 2 millones de seguidores ó
               tienen más de 500 mil vistas y más de 1 millón de likes.
        2. Su última publicación fue estrictamente en el mismo año que el ingresado (anio).
    Args:
        anio (str): Año a evaluar. Debe ser de longitud 4 y convertido a int, debe ser positivo.
        c1, c2 (dict): Diccionarios que representan a los creadores de contenido.
    Returns:
        str: Nombres y ratings de los creadores que cumplen todas las condiciones, separados por comas.
                Ejemplo: "Ana 1.34, Luis 1.3".
                    Note que, en ejemplo, Ana y Luis son los nombres de los creadores de contenido y
                        1.34 y 1.3 son sus respectivos ratings.
             Si ningun creador cumple, retorna el string: "Ninguno".
    """
```

Ejemplo de solución:

```
    assert isinstance(anio, str), "Error: El parámetro 'anio' debe ser de tipo str."
    assert isinstance(c1, dict), "Error: El creador c1 debe ser de tipo dict."
    assert isinstance(c2, dict), "Error: El creador c2 debe ser de tipo dict."
    assert len(anio) == 4, "Error: El año debe tener 4 dígitos en formato YYYY."
    assert int(anio) > 0, "Error: El año debe ser positivo."

    resultado = ""

    SEGUIDORES_MIN = 2000_000
    VISITAS_MIN = 500_000
    LIKES_MIN = 1000_000

    rating1 = calcular_rating_creador(c1)
    rating2 = calcular_rating_creador(c2)

    if (
        c1["pais"] == "Colombia"
        and anio in c1["fecha_ultima_publicacion"]
        and (c1["seguidores"] > SEGUIDORES_MIN or (c1["vistas"] > VISITAS_MIN and c1["likes"] > LIKES_MIN))
    ):
        resultado = f'{c1["nombre"]} {rating1}'

    if (
        c2["pais"] == "Colombia"
        and anio in c2["fecha_ultima_publicacion"]
        and (c2["seguidores"] > SEGUIDORES_MIN or (c2["vistas"] > VISITAS_MIN and c2["likes"] > LIKES_MIN))
    ):
        if resultado != "":
            resultado += ", "
        resultado += f'{c2["nombre"]} {rating2}'

    if resultado == "":
        resultado = "Ninguno"

    return resultado
```

Errores comunes:

- Violar la buena práctica BP-6. Tema que se recomendó explícitamente estudiar.
- No considerar nunca el país (`"Colombia"`). Esto indica problemas de comprensión de lectura.
- No considerar nunca a los seguidores, vistas y/o likes.
- No producir un resultado del tipo `str` (Ej: un diccionario), `"Ninguno"`, resultados sin separación por comas o resultados que concatenan: `"Ninguno"` a el o los nombre(s) y rating(s).
- No validar correctamente `anio` (su longitud, que sea positivo, o verificarlo como si fuese un parámetro del tipo `int`) o nunca validar la condición:

```
2. Su última publicación fue estrictamente en el mismo año que el ingresado (anio).
```

- Intentar validar la condición del año de la forma errónea:

```
anio == c1["fecha_ultima_publicacion"]
```

  - Esto nunca funcionaría, porque el formato de `anio` es: `"YYYY"`, mientras el de `["fecha_ultima_publicacion"]` es: `"YYYY-MM-DD"`. Era más fácil usar `in`, lo que se recomendó explícitamente estudiar y además se mostraron ejemplos en diapositivas durante el parcial.

- Intentar validar la condición del año de la forma errónea:

```
anio == c1["fecha_ultima_publicacion"][0] # o similares
```

  - Esto nunca funcionaría, porque en la posición `0` o similares en `["fecha_ultima_publicacion"]` es: solo una letra entre las posicionadas en el formato: `"YYYY-MM-DD"`. Era más fácil usar `in`.

- No retornar nada.
- Usar variables nunca antes declaradas. Esto indica una clara incomprensión del curso.
- Usar un algoritmo que no cumple con seleccionar el o los creadores apropiados. Típicamente, retornando prematuramente un solo creador, a pesar de que el otro también cumple.

Pregunta 3 (30 pts). Implemente en Python la función `buscar_creador_mas_activo`, tal y como se describe en la documentación (siguiente página). Note que solo se usan dos creadores de contenido, en lugar de cuatro, para facilitarle la escritura de la función y que esta no sea muy extensa. No debe crear doctests y solo se le piden crear **una sola** aserción muy específica (ver la lista a continuación):

Lista de chequeo:

- ☐ La función incluye una sola aserción que verifica la condición de rango de `minimo_rating`.
- ☐ La función **no** usa funciones como `min()` y `max()`.
- ☐ La función usa condicionales y **no** usa el operador ternario.
- ☐ Se usa la comparación lexicográfica que aplica a fechas en el formato `'YYYY-MM-DD'`.

La siguiente, es la rúbrica de calificación que se usará para calificar la pregunta 3:

Descripción	Excelente	Bien	Regular	Insuficiente
La implementación de la función cumple con lo solicitado: 30 pts si cumple a cabalidad ya que el algoritmo es correcto, cumple con todas las buenas prácticas del curso (BP-1 a BP-23) y con toda la lista de chequeo. 20 pts si el algoritmo es parcialmente correcto y falla en detalles muy menores y/o se incumple hasta una sola buena práctica del curso (BP-1 a BP-23) o un solo ítem de la lista de chequeo. 10 pts si el algoritmo es parcialmente correcto con errores menores y/o se incumple hasta dos buenas prácticas del curso (BP-1 a BP-23) o dos ítems de la lista de chequeo. 0 pts si el algoritmo tiene errores graves y/o se incumple más de dos buenas prácticas del curso (BP-1 a BP-23) o más de dos ítems de la lista de chequeo.				
Observaciones del profesor:				

```python
def buscar_creador_mas_activo(
    fecha_inicio: str,
    fecha_fin: str,
    minimo_rating: float,
    c1: dict,
    c2: dict
) -> dict:
    """
    Busca al creador más activo. Este es aquel con la fecha de última publicación
    (fecha_ultima_publicacion) más reciente, dentro del rango inclusivo: [fecha_inicio, fecha_fin] y
    cuyo rating supera al parámetro minimo_rating.
    Args:
        fecha_inicio (str): Fecha inicial del rango. Está en formato "YYYY-MM-DD".
        fecha_fin (str): Fecha final del rango. Está en formato "YYYY-MM-DD".
        minimo_rating (float): Rating que debe superar el creador. Valor entre 0.0 y 100.0.
        c1, c2 (dict): Diccionarios que representan a los creadores de contenido.
    Returns:
        dict: Diccionario con las dos llaves 'nombre' y 'fecha', cuyos valores asociados
                serán el nombre y fecha de última publicación del creador más activo.
                Ejemplo: {'nombre': 'Joshua', 'fecha': '2025-01-28'}
                    Note que, en ejemplo, Joshua es el nombre del creador más activo y
                        2025-01-28 es la fecha de última publicación del mismo.
                Si hay empate de fecha, se retorna quien tenga más seguidores.
                    Si el empate persiste, se retorna el primero en el orden (c1, c2).
                        Si ningún creador cumple con los criterios, se retorna un diccionario vacío ({}).
    """
```

Ejemplo de solución:

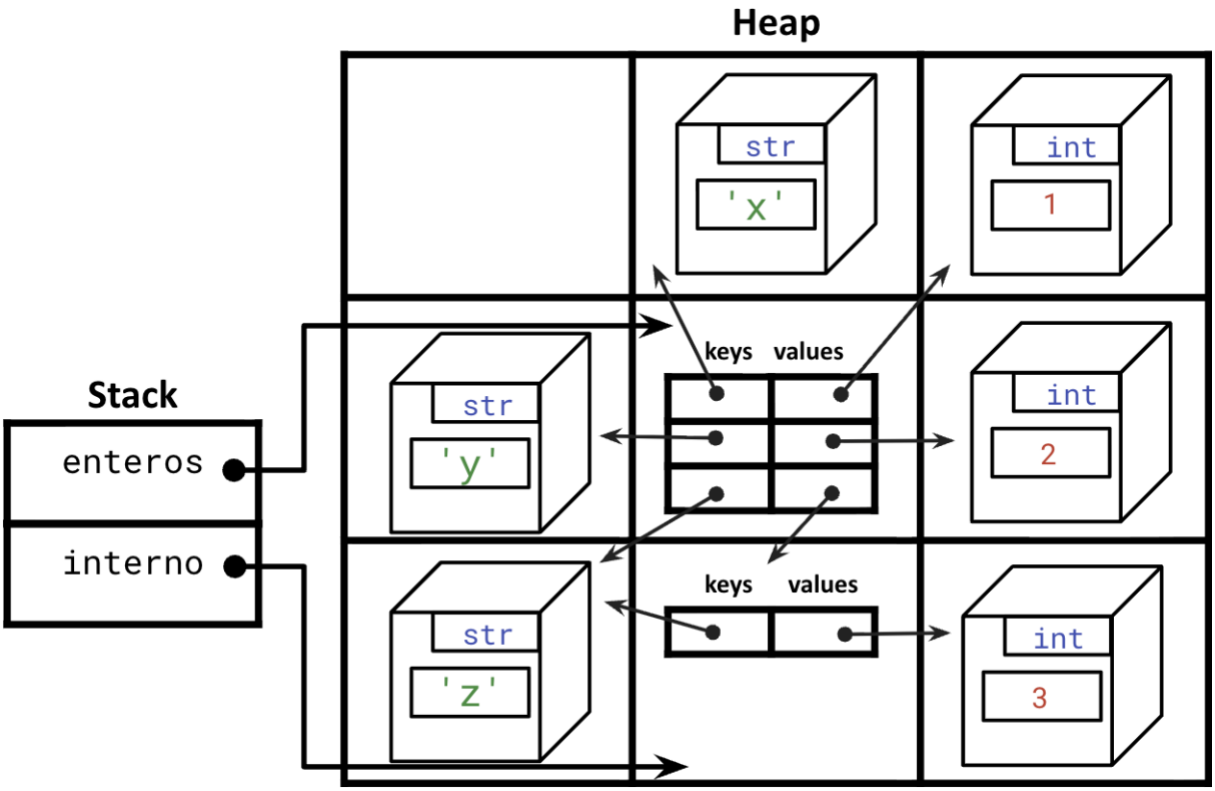
```python
    assert 0 <= minimo_rating <= 100, "Error: El rating mínimo debe ser un valor entre 0 y 100."

    resultado = None
    creador_buscado = {}
    fecha_buscada = ""
    max_seguidores = 0

    rating1 = calcular_rating_creador(c1)
    rating2 = calcular_rating_creador(c2)

    if fecha_inicio <= c1["fecha_ultima_publicacion"] <= fecha_fin and rating1 > minimo_rating:
        creador_buscado = c1
        fecha_buscada = c1["fecha_ultima_publicacion"]
        max_seguidores = c1["seguidores"]

    if fecha_inicio <= c2["fecha_ultima_publicacion"] <= fecha_fin and rating2 > minimo_rating:
        if (not creador_buscado or c2["fecha_ultima_publicacion"] > fecha_buscada or
            (c2["fecha_ultima_publicacion"] == fecha_buscada and c2["seguidores"] > max_seguidores)):
            creador_buscado = c2
            fecha_buscada = c2["fecha_ultima_publicacion"]

    if creador_buscado:
        resultado = {"nombre": creador_buscado["nombre"], "fecha": fecha_buscada}

    return resultado
```

Errores comunes:

- Violar la buena práctica BP-18. Tema que se recomendó explícitamente estudiar.
- No validar de forma inclusiva el rango de `fecha_ultima_publicacion`. Esto indica problemas de comprensión de lectura.
- Usar un campo del diccionario del creador que no existe (Ej: `["fecha"]`). Esto indica problemas de comprensión de lectura.
- No validar el rango (`Valor entre 0.0 y 100.0`) de `minimo_rating`. Esto indica problemas de comprensión de lectura.
- No validar que el creador debe superar el `minimo_rating`. Esto indica problemas de comprensión de lectura.
- Nunca manejar el caso de empate, considerando el número de seguidores. Tema que se recomendó explícitamente estudiar y se entregaron doctests para que se pudiese practicar con N2-PROY.
- No producir un resultado del tipo `dict` (Ej: un string).
- Concatenar diccionarios y/o diccionarios con strings. Esto indica una clara incomprensión del curso.
- Comparar diccionarios completos directamente con strings. Esto indica una clara incomprensión del curso.
- No retornar nada.
- Usar variables nunca antes declaradas. Esto indica una clara incomprensión del curso.
- Usar un algoritmo que no cumple con seleccionar el creador apropiado, típicamente retornando el primero antes de verificar correctamente al segundo o retornando al segundo, sin verificar que cumpla con todas las condiciones (solo al descartar que el primero cumpla).

Pregunta 4 (10 pts). Esta es una pregunta de **selección única**. Marque (rellene el círculo) de la única opción con las instrucciones en Python que, al ser ejecutadas exitosamente, producen el estado del Stack y del Heap que se muestra en la siguiente imagen.

Nota: Por simplicidad, en el Stack solo se incluye el nombre de variables.



○	`enteros = {'x':1, 'y':2, 'z': interno}` `interno = {'z':3}`
○	`enteros = {'x':1, 'y':2, 'z':{'z':3}}`
○	`enteros = {'x':1, 'y':2, 'z':3}` `interno = {'z':3}`

X	`interno = {'z':3}`
	`enteros = {'x':1, 'y':2}`
	`enteros['z'] = interno`

○	`interno = {'z':3}` `enteros = {'x':1, 'y':2, 'z':3}`

○	`interno = {'z': 3}`
	`enteros = {'x': 1, 'y': 2}`
	`interno['z'] = enteros`

○	`interno = {'z': 3}`
	`enteros = {'x': 1, 'y': 2}`
	`enteros['z'] = {'z': 3}`

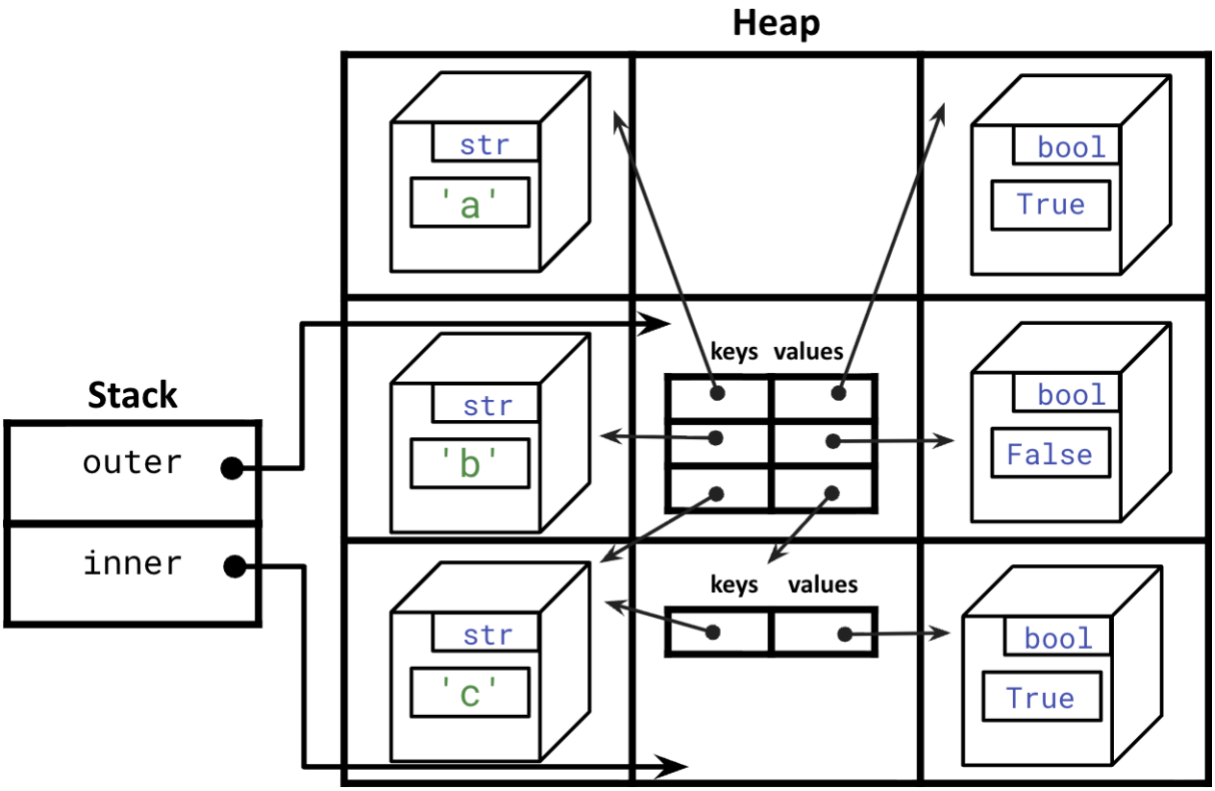
○	`interno = {'z': 3}`
	`enteros = {'x': 1, 'y': 2}`
	`enteros = interno`

○	`interno = {z:3}`
	`enteros = {x:1; y:2}`
	`enteros[z] = interno`

○	`interno = {'x':1, 'y':2, 'z':{'z':3}}`

Pregunta 4 (10 pts). Esta es una pregunta de **selección única**. Marque (rellene el círculo) de la única opción con las instrucciones en Python que, al ser ejecutadas exitosamente, producen el estado del Stack y del Heap que se muestra en la siguiente imagen.

Nota: Por simplicidad, en el Stack solo se incluye el nombre de variables.



○	`externo = {'a':True, 'b':False, 'c': interno}` `interno = {'c':True}`
○	`externo = {'a':True, 'b':False, 'c':{'c':True}}`
○	`externo = {'a':True, 'b':False, 'c':True}` `interno = {'c':True}`
○	`interno = {'c':True}` `externo = {'a':True, 'b':False, 'c':True}`

○	`interno = {'c': True}`
	`externo = {'a': True, 'b': False}`
	`interno['c'] = externo`

○	`interno = {'c': True}`
	`externo = {'a': True, 'b': False}`
	`externo['c'] = {'c': True}`

X	`interno = {'c':True}`
	`externo = {'a':True, 'b':False}`
	`externo['c'] = interno`

○	`interno = {'c': True}`
	`externo = {'a': True, 'b': False}`
	`externo = interno`

○	`interno = {c:True}`
	`externo = {a:True; b:False}`
	`externo[c] = interno`

○	`interno = {'a':True, 'b':False, 'c':{'c':True}}`