Status	Finished
Started	Friday, 22 November 2024, 11:00 AM
Completed	Friday, 22 November 2024, 11:04 AM
Duration	4 mins 13 secs
Marks	6.00/6.00
Grade	**10.00** out of 10.00 (**100**%)

Information

Entrada de datos

Un algoritmo describe la solución de un problema en función de los datos necesarios para representar un caso concreto del problema y de los pasos necesarios para obtener el resultado deseado. El tratamiento de la información plantea problemas cuyo propósito es hallar un resultado desconocido a partir de datos conocidos.

Para que un ordenador resuelva un caso de un problema, teniendo el programa para hacerlo, es necesario proporcionarle los datos del caso; esto se puede hacer mediante instrucciones de entrada. Véase el siguiente programa:

```
print("Hola ¿Cómo te llamas?")
name = input()
print("Hola", name)
```

El resultado de la ejecución de este programa, suponiendo que el usuario introdujera por teclado el nombre Pedro, sería el siguiente:

```
Hola ¿Cómo te llamas?
Pedro
Hola Pedro
```

El programa usa dos veces la instrucción print, que es una **instrucción de salida**, es decir, una instrucción para dar información, y aparece una instrucción nueva, input, que es una **instrucción de entrada**, una instrucción para recoger datos que van a ser usados por el programa.

También aparece otro elemento, *name*, que no es una instrucción, sino el nombre de una **variable**. Una variable se puede ver como una etiqueta que se usa para hacer referencia a un dato manejado por el programa.

El valor del dato referenciado por una variable puede ser distinto en diferentes ejecuciones del programa (supóngase que el programa anterior se ejecuta para otro usuario llamado Juan) y puede, incluso, variar durante la ejecución del programa (de ahí el nombre de variable). Esta capacidad de variación de los datos es lo que permite que un programa resuelva diferentes casos de un mismo problema.

El efecto de la ejecución de la instrucción de entrada input es que el programa espera a que el usuario teclee el valor del dato deseado, terminando con la tecla ↵ (denominada intro o entrar, en inglés *enter*). Una vez que lo hace, se recoge ese valor y, tal como indica el código mostrado más arriba, se asigna a la variable que está a la izquierda del símbolo de **asignación** **=**. Como el programa se va a detener hasta que el usuario introduzca el valor de un dato, lo normal es que se le indique justo antes con un mensaje qué dato es el que se espera que introduzca; esto se hace en el ejemplo anterior con la primera instrucción print, pero se puede abreviar usando la propia instrucción input:

```
name = input("Hola ¿Cómo te llamas? ")
print("Hola", name)
```

El resultado de la ejecución de este programa es:

```
Hola ¿Cómo te llamas? Pedro
Hola Pedro
```

Nótese que en este caso la entrada del valor se espera en la misma línea donde se muestra el mensaje que lo solicita, de ahí que en la instrucción input hayamos añadido un espacio después del '?', para evitar que el nombre que teclee el usuario se visualice pegado al '?' (si el usuario tecleara un espacio antes del nombre, ese espacio formaría parte del contenido de la variable name, cosa que en principio no queremos).

En ambas versiones del programa, la última instrucción muestra un saludo personalizado que incluye el valor del dato introducido por el usuario:

```
print("Hola", name)
```

name no se encierra entre comillas; las comillas sirven para indicar un valor literal tal cual (la palabra Hola en el ejemplo). En el caso de la variable name, lo que se quiere mostrar no es la palabra name, sino el dato referenciado por la variable (Pedro en el ejemplo).

Question 1

Complete

Mark 1.00 out of 1.00

¿Qué instrucción se usa en Python para recoger datos tecleados por el usuario?

Answer: input()

Information

Los nombres de las variables

Los programas usan **variables** para hacer referencia a los datos del problema que resuelven: datos de inicio, que varían de un caso del problema a otro; datos intermedios, generados durante la resolución del problema, y que pueden variar a medida que se ejecuta el programa; y datos finales, con los resultados que conforman la solución del problema.

Las variables tienen un **nombre** y hacen referencia al **valor** de un dato que se almacena en la memoria del ordenador accesible a la actual ejecución del programa.

Las **reglas léxicas** de Python requieren que los nombres de las variables empiecen por una letra mayúscula (A··Z), o minúscula (a··z)), o un guion bajo/subrayado (_), pudiendo continuar con cualquier combinación de letras, guiones bajos y dígitos (0··9):

```
nombre
NOMBRE
_nombre_1
fecha_de_nacimiento
xASFasDasd24_d
```

Las **reglas de estilo** de Python establecen que los nombres de las variables deben escribirse en minúsculas, separando por guiones bajos sus componentes cuando se trate de nombres compuestos, y ser descriptivos respecto al dato que representan:

```
nombre
~~NOMBRE~~
_nombre_1
fecha_de_nacimiento
~~xASFasDasd24_d~~
```

Question 2

Complete

Mark 1.00 out of 1.00

¿Cuáles de los siguientes identificadores cumplen tanto las reglas léxicas como las reglas de estilo para ser el nombre de una variable en Python?

Select one or more:

- [x] _nombre_1
- [x] fecha_de_nacimiento
- [ ] 12_monos
- [x] xASFasDasd24_d
- [x] nombre
- [x] NOMBRE

Information

Los valores de las variables

Los programas manejan datos usando variables. Las variables tienen un **nombre** (**identificador**) y hacen referencia (contienen) al **valor** de un dato. Para asociar (**asignar**) un valor a una variable, se usa una **instrucción de asignación**, caracterizada por el símbolo =.

```
name = input()           # asigna a name un valor introducido por el usuario
greeting = "Hola, mundo" # asigna a greeting un texto literal
```

Podemos definir un **dato** como "una representación simbólica (numérica, alfabética, algorítmica, espacial, etc.) de un atributo cualitativo o una magnitud cuantitativa". Existen muchos tipos de datos diferentes. En el ejemplo anterior, a las variables name y greeting se les asignan valores que representan un pequeño texto. Los valores que representan texto se conocen como **strings**. Una string es simplemente una **secuencia de caracteres**. Las strings pueden usarse para representar cosas como nombres, notas, observaciones, etc.

Otros posibles tipos de datos que se pueden asociar una variable son los números; por ejemplo, números **enteros** (sin parte fraccionaria):

```
age = 21
```

números **reales** (que incluyen parte fraccionaria, antes de la cual se ha de usar el punto, no la coma):

```
temperature = 36.5
```

números **complejos** (con parte real y parte imaginaria; la unidad imaginaria se representa por **j**):

```
complex_number = -34+5.3j
```

También se pueden usar valores **booleanos**, que se representan con **True** (verdadero) o **False** (falso):

```
bool_data = True
```

Los valores booleanos pueden ser el resultado de una comparación u otro tipo de operación:

```
bool_data = 8 > 5 # se asigna el valor False, resultado de la expresión 8 > 5
```

Nótese que solo los **literales** de valores de tipo texto (strings) se representan encerrados entre comillas.

Las strings y los números (incluyendo los booleanos) son tipos **básicos** en Python. Python predefine también otros tipos de datos (tuplas, listas, diccionarios, ...) o los incorpora a través de librerías externas, aparte de dar la posibilidad al programador de definir nuevos tipos según sus necesidades.

Information

Asignación

Los programas manejan datos usando variables. Las variables tienen un **nombre** y hacen referencia al **valor** de un dato. Para asociar un (nuevo) valor con una variable, se usa una **instrucción de asignación**, caracterizada por el símbolo **=**. A la izquierda del **=** se indica una variable y a su derecha un valor (literal o a obtener de una cálculo u operación).

En la **ejecución** de una instrucción de asignación, **una vez que se obtiene el valor resultante del lado derecho** del **=**, se le asigna a continuación a la variable del lado izquierdo, sustituyendo al valor que contuviera previamente.

A una variable se le puede asignar:

- El resultado de una operación de entrada

```
var1 = input()
```

- Un valor expresado literalmente (recuérdese que solo las strings literales se encierran entre comillas)

```
var2 = 35
var3 = "Hola"
```

- Un cálculo (se asigna el valor resultante)

```
var4 = 7 * 5 + 18 # a var4 se le asigna el valor 53
```

- Otra variable (para tipos básicos de Python, a los efectos, se asigna una copia del valor de la variable de la derecha)

```
var5 = var4
```

- Las expresiones pueden incluir variables

```
var6 = var4 + var5 - 10 # el valor asignado a var6 es el resultado 96
```

- La propia variable a la que se asigna el resultado de una expresión puede estar incluida en la expresión (el mismo nombre de variable puede aparecer a ambos lados del **=**, pero significa cosas distintas: en el lado derecho se toma su valor para realizar un cálculo, mientras que en el lado izquierdo indica el destino al que vamos a asignar ese valor)

```
var6 = var6 + 1 # el valor asignado a var6 es 97
```

- Si la variable forma parte de la expresión, se puede abreviar en algunos casos frecuentes

```
var6 += 1 # equivale a var6 = var6 + 1
var6 *= 2 # equivale a var6 = var6 * 2
```

Una vez vistos los ejemplos, podemos resumir diciendo que una instrucción de asignación se compone siempre de un símbolo = con una **expresión** a su derecha (que puede ser muy simple, como un literal o una variable, o contener operaciones), cuyo valor resultante se asigna a la variable de la izquierda.

Question 3

Complete

Mark 1.00 out of 1.00

¿Qué valor referencia la variable b después de la siguiente secuencia de asignaciones?

a = 5
b = 8
c = a + b
b += c - a

Answer: 16

Question 4

Complete

Mark 1.00 out of 1.00

¿Qué valor referencia la variable b después de la siguiente secuencia de asignaciones?

```
a = 5
b = 8
c = a + b
b += a - b
```

Answer: 5

Information

Asignación múltiple

Los programas manejan datos usando variables. Las variables tienen un **nombre** y hacen referencia al **valor** de un dato. Para asociar un valor con una variable, se usa una **instrucción de asignación,** representada por el símbolo =. El funcionamiento de la asignación es: primero se evalúa la expresión que está a la derecha del símbolo de asignación y, luego, el resultado de esa evaluación se asigna a la variable que está a la izquierda del símbolo de asignación.

```
var1 = 35
```

Además de lo dicho, Python permite asignar múltiples valores a múltiples variables en una sola instrucción de asignación:

```
a, b, c = 1, 2, 3
```

A la derecha de la asignación debe haber tantas expresiones, separadas por comas, como variables haya a la izquierda de la asignación. Los valores de las expresiones se asignan en paralelo ("simultáneamente") de izquierda a derecha a las variables correspondientes: el primer valor a la primera variable, el segundo valor a la segunda variable, etc. En el ejemplo anterior, el valor 1 se asigna a la variable a, el valor 2 a la variable b y el valor 3 a la variable c.

La asignación múltiple permite, entre otras cosas, intercambiar cómodamente los valores de dos variables con una sola instrucción (en el ejemplo a recibe el anterior valor de b, y simultáneamente, b recibe el anterior valor de a).

```
a, b = b, a
```

Esto mismo, en muchos otros lenguajes, requeriría tres instrucciones y una variable adicional:

```
aux = a
a = b
b = aux
```

La variable adicional (aux en el ejemplo) sirve para guardar el valor de la primera variable que se va a cambiar, de forma que no se pierda y pueda asignársele luego a la otra.

Question 5

Complete

Mark 1.00 out of 1.00

¿Qué valor referencia la variable b después de la siguiente asignación?

```
a, c, b = 21, 12, 34
```

Answer: 34

Information

Entrada de valores no textuales

La instrucción input recoge los datos, sean del tipo que sean, como una secuencia de caracteres (string), tal como se introducen por teclado.

```
name = input("¿Cómo te llamas? ")
```

Si la secuencia de caracteres introducidos se puede interpretar como un valor válido de otro tipo primitivo (número entero, número real, número complejo), hay que indicar que se quiere hacer esa **conversión** (el caso bool es especial):

```
text = input("Dame una string formada por una secuencia de dígitos: ")
integer_number = int(input("Dame un número entero: "))
real_number = float(input("Dame un número real: "))
complex_number = complex(input("Dame un número complejo: "))
bool_value = bool(int(input("Dame un booleano (0->False, 1->True): ")))
```

Resultado de la ejecución (se resaltan los valores introducidos por el usuario):

```
Dame una string formada por una secuencia de dígitos: 23289
Dame un número entero: 23289
Dame un número real: 23289.0
Dame un número complejo: 14+23j
Dame un booleano (0->False, 1->True): 0
```

Nótese el uso de **int**, **float** o **complex** para indicar la interpretación requerida. Las strings (secuencias de caracteres) se identifican como **str**, pero sería redundante indicarlo en una instrucción de entrada, dado que ese es el tipo por omisión de los datos entrados por teclado. El caso de **bool()** es especial como veremos, no nos serviría teclear False para indicar este valor, de ahí que recurramos a los int 0 y 1 para luego convertir a bool.

En caso de que la secuencia de caracteres tecleada por el usuario no pueda interpretarse como un valor del tipo requerido, se produce un error.

Cada tipo de datos usa una representación interna en memoria diferente como secuencia de bits (ceros y unos), en cada caso usando una codificación diferente. Incluso en casos como los referenciados por las variables text e integer_number en el ejemplo, que responden a la misma secuencia de caracteres de entrada: en el primer caso cada caracter (de "cifra") se representa en binario por separado de acuerdo con su codificación estándar (p.e. cada '2' se podría representar por una secuencia 00110010), mientras que en el segundo se convierte a un valor numérico binario equivalente (en notación denominada *complemento a dos*), concretamente 0··0101101011111001 (23289 en binario).

El nombre del tipo *float*, que usamos para los números reales, se debe a que usa una representación interna llamada *floating point* (coma flotante), análoga a la notación científica de nuestras calculadoras (que usa un exponente), de forma que la coma (el punto) no está siempre justo a la izquierda de la parte fraccionaria (*fixed point*), sino que "flota" en combinación con el exponente, p.e. 123.45 (escrito en coma fija) podemos imaginar que internamente se representa como $0.12345 \cdot 10^3$ (en realidad, algo similar pero adaptado a binario), o sea, el punto ha "flotado" hasta justo antes de la primera cifra distinta de cero adecuándose al mismo tiempo el exponente. De esta manera, el número queda representado por un par de enteros, en el ejemplo 12345 (la *mantisa*) y 3 (el exponente). El tamaño de la mantisa (y del exponente) es limitado, por lo que en muchos casos solo podemos representar un número real de forma aproximada.

Question 6

Complete

Mark 1.00 out of 1.00

Relacione cada tipo de datos con su identificador en Python.

Secuencia de caracteres	str
Número real	float
Número entero	int
Valor booleano	bool
Número complejo	complex