Introducción

Antes de empezar con las lecciones es muy importante que cree un directorio (carpeta) en su repositorio, el que venimos trabajando el clase, en este caso como son las segundas lecciones solo deberá entrar al directorio **lecciones** con el comando **cd lecciones**, una vez que estamos en directorio **lecciones** creamos nuestro directorio para resolver todas las lecciones de esta unidad, deberá crearlo con el comando **mkdir lecciones_02**, donde creará todos los programas (los archivos con extensión .py) para realizar las lecciones.

Resumen de pasos:
cd lecciones (para acceder al directorio).
mkdir lecciones_02 (para crear el directorio para estas lecciones).
cd lecciones_02 (para acceder al directorio).

Pensamiento Profundo

> ***"Muy bien,"*** *dijo la computadora, y se sumió en silencio nuevamente. Los dos hombres se inquietaron. La tensión era insoportable.*
> ***"Realmente no les va a gustar,"*** *observó Pensamiento Profundo.*
> *"¡Díganos!"*
> ***"Muy bien,"*** *dijo Pensamiento Profundo.* ***"La Respuesta a la Gran Pregunta…"***
> ***"¡Sí…!"***
> ***"De la Vida, el Universo y Todo…"*** *dijo Pensamiento Profundo.*
> *"¡Sí…!"*
> ***"Es…"*** *dijo Pensamiento Profundo, y hizo una pausa.*
> *"¡Sí…!"*
> ***"Es…"***
> *"¡¿Sí…!!!…?"*
> ***"Cuarenta y dos,"*** *dijo Pensamiento Profundo, con infinita majestuosidad y calma.*
>
> *— La Guía del Viajero Intergaláctico, Douglas Adams*

En `pensamiento_profundo.py`, implementa un programa que solicite al usuario la respuesta a la Gran Pregunta de la Vida, el Universo y Todo, mostrando `Sí` si el usuario ingresa `42` o (sin importar mayúsculas/minúsculas) `cuarenta y dos` o `cuarentaidos`. De lo contrario, muestra `No`.

Consejos

- ¡No necesitas convertir la entrada del usuario a un `int` si verificas la igualdad con `"42"`, un `str`, en lugar de `42`, un `int`!
- Está bien si tu salida o la entrada del usuario se extiende a múltiples líneas.

Demostración

```
$ python pensamiento_profundo.py
¿Cuál es la respuesta a la gran pregunta sobre la vida, el universo y todo lo demás? 42
Sí
$ python pensamiento_profundo.py
¿Cuál es la Respuesta a la Gran Pregunta de la Vida, el Universo y Todo? cuarenta y dos
Sí
$ python pensamiento_profundo.py
¿Cuál es la Respuesta a la Gran Pregunta de la Vida, el Universo y Todo? cuarentaidos
Sí
$
```

Antes de Comenzar

Ejecuta:

mkdir pensamiento_profundo

para hacer una carpeta llamada `pensamiento_profundo` en tu espacio de código.

Entonces ejecuta:

cd pensamiento_profundo

para cambiar directorios a esa carpeta. Ahora deberías ver tu prompt de terminal como `pensamiento_profundo/ $`. Ahora puedes ejecutar:

code pensamiento_profundo.py

para hacer un archivo llamado `pensamiento_profundo.py` donde escribirás tu programa.

Cómo Probar

Así es como probar tu código manualmente:

- Ejecuta tu programa con `python pensamiento_profundo.py`. Escribe `42` y presiona Enter. Tu programa debería mostrar:

  Sí

- Ejecuta tu programa con `python pensamiento_profundo.py`. Escribe `Cuarenta Y Dos` y presiona Enter. Tu programa debería mostrar:

  Sí

- Ejecuta tu programa con `python pensamiento_profundo.py`. Escribe `cuarentaidos` y presiona Enter. Tu programa debería mostrar:

  Síl

- Ejecuta tu programa con `python pensamiento_profundo.py`. Escribe `50` y presiona Enter. Tu programa debería mostrar:

  No

Asegúrate de variar las mayúsculas y minúsculas de tu entrada y "accidentalmente" agregar espacios en cualquier lado de tu entrada antes de presionar enter. Tu programa debería comportarse como se espera, sin importar mayúsculas/minúsculas y espacios.

Banco Federal de Ahorros

Kramer visita un banco que promete dar $100 a cualquier persona que no sea saludada con un "hola". En su lugar, Kramer es saludado con un "hey", lo cual él insiste que no es un "hola", y por eso pide $100. El gerente del banco propone un compromiso: "Recibiste un saludo que comienza con 'h', ¿qué te parecen $20?" Kramer acepta.

En un archivo llamado `banco.py`, implementa un programa que le pida al usuario un saludo. Si el saludo comienza con "hola", devuelve `$0`. Si el saludo comienza con "h" (pero no "hola"), devuelve `$20`. De lo contrario, devuelve `$100`. Ignora cualquier espacio en blanco al inicio del saludo del usuario, y trata el saludo del usuario sin distinguir mayúsculas y minúsculas.

Pistas

- Recuerda que un `str` viene con varios métodos, según docs.python.org/3/library/stdtypes.html#string-methods.
- Asegúrate de dar $0 no solo para "hola" sino también "hola amigo", "hola, Newman", y similares.

Antes de Comenzar

Ejecuta

mkdir banco

para crear una carpeta llamada `banco` en tu codespace.

Después ejecuta

cd banco

para cambiar directorios a esa carpeta. Ahora deberías ver tu prompt de terminal como `banco/ $`. Ahora puedes ejecutar

code banco.py

para crear un archivo llamado `banco.py` donde escribirás tu programa.

Cómo Probar

Aquí está cómo probar tu código manualmente:

- Ejecuta tu programa con `python banco.py`. Escribe `Hola` y presiona Enter. Tu programa debería mostrar:

$0

- Ejecuta tu programa con `python banco.py`. Escribe `Hola, Newman` y presiona Enter. Tu programa debería mostrar:

  $0

- Ejecuta tu programa con `python banco.py`. Escribe `¿Cómo estás?` y presiona Enter. Tu programa debería mostrar:

  $20

- Ejecuta tu programa con `python banco.py`. Escribe `¿Qué está pasando?` y presiona Enter. Tu programa debería mostrar:

  $100

Extensiones de Archivo

Aunque Linux, Windows y macOS a veces las ocultan, la mayoría de los archivos tienen extensiones de archivo, un sufijo que comienza con un punto (.) al final de su nombre. Por ejemplo, los nombres de archivos para GIFs terminan con `.gif`, y los nombres de archivos para JPEGs terminan con `.jpg` o `.jpeg`. Cuando haces doble clic en un archivo para abrirlo, tu computadora usa su extensión de archivo para determinar qué programa ejecutar.

Los navegadores web, por el contrario, se basan en tipos de medios, anteriormente conocidos como tipos MIME, para determinar cómo mostrar archivos que se encuentran en la web. Cuando descargas un archivo desde un servidor web, ese servidor envía un encabezado HTTP, junto con el archivo en sí, indicando el tipo de medio del archivo. Por ejemplo, el tipo de medio para un GIF es `image/gif`, y el tipo de medio para un JPEG es `image/jpeg`. Para determinar el tipo de medio de un archivo, un servidor web típicamente examina la extensión del archivo, mapeando una con la otra.

Consulta developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types para tipos comunes.

En un archivo llamado `extensiones.py`, implementa un programa que solicite al usuario el nombre de un archivo y luego muestre el tipo de medio de ese archivo si el nombre del archivo termina, sin distinción de mayúsculas y minúsculas, en cualquiera de estos sufijos:

- `.gif`
- `.jpg`
- `.jpeg`
- `.png`
- `.pdf`
- `.txt`
- `.zip`

Si el nombre del archivo termina con algún otro sufijo o no tiene sufijo en absoluto, muestra `application/octet-stream` en su lugar, que es un valor predeterminado común.

Pistas

- Recuerda que un `str` viene con bastantes métodos, según docs.python.org/3/library/stdtypes.html#string-methods.

Antes de Comenzar

Ejecuta

mkdir extensiones

para crear una carpeta llamada `extensiones` en tu codespace.

Luego ejecuta

cd extensiones

para cambiar directorios a esa carpeta. Ahora deberías ver tu prompt del terminal como `extensiones/ $`. Ahora puedes ejecutar

code extensiones.py

para crear un archivo llamado `extensiones.py` donde escribirás tu programa.

Cómo Probar

Aquí se explica cómo probar tu código manualmente:

- Ejecuta tu programa con `python extensiones.py`. Escribe `feliz.jpg` y presiona Enter. Tu programa debería mostrar:

  image/jpeg

- Ejecuta tu programa con `python extensiones.py`. Escribe `documento.pdf` y presiona Enter. Tu programa debería mostrar:

  application/pdf

Asegúrate de probar cada uno de los otros formatos de archivo, varía las mayúsculas y minúsculas de tu entrada, y "accidentalmente" agrega espacios a ambos lados de tu entrada antes de presionar enter. Tu programa debería comportarse como se esperaba, sin distinción de mayúsculas y minúsculas y espacios.

Intérprete Matemático

Python ya soporta matemáticas, por lo que puedes escribir código para sumar, restar, multiplicar o dividir valores e incluso variables. Pero vamos a escribir un programa que permita a los usuarios hacer matemáticas, incluso sin conocer Python.

En un archivo llamado `interprete.py`, implementa un programa que solicite al usuario una expresión aritmética y luego calcule y muestre el resultado como un valor de punto flotante formateado con un decimal. Asume que la entrada del usuario estará formateada como `x y z`, con un espacio entre `x` e `y` y un espacio entre `y` y `z`, donde:

- `x` es un entero
- `y` es `+`, `-`, `*`, o `/`
- `z` es un entero

Por ejemplo, si el usuario ingresa `1 + 1`, tu programa debe mostrar `2.0`. Asume que, si `y` es `/`, entonces `z` no será `0`.

Ten en cuenta que, así como `python` es un intérprete para Python, tu `interprete.py` será un intérprete para matemáticas.

Pistas

Recuerda que un `str` viene con varios métodos, según docs.python.org/3/library/stdtypes.html#string-methods, incluyendo `split`, que separa un `str` en una secuencia de valores, todos los cuales pueden ser asignados a variables de una vez. Por ejemplo, si `expresion` es un `str` como `1 + 1`, entonces:

x, y, z = expresion.split(" ")

asignará `1` a `x`, `+` a `y`, y `1` a `z`.

Antes de Comenzar

Ejecuta:

mkdir interprete

para crear una carpeta llamada `interprete` en tu espacio de código.

Entonces ejecuta:

cd interprete

para cambiar directorios a esa carpeta. Ahora deberías ver el prompt de tu terminal como `interprete/ $`. Ahora puedes ejecutar:

code interprete.py

para crear un archivo llamado `interprete.py` donde escribirás tu programa.

Cómo Probar

Aquí tienes cómo probar tu código manualmente:

- Ejecuta tu programa con `python interprete.py`. Escribe `1 + 1` y presiona Enter. Tu programa debe mostrar:

  2.0

- Ejecuta tu programa con `python interprete.py`. Escribe `2 - 3` y presiona Enter. Tu programa debe mostrar:

  -1.0

- Ejecuta tu programa con `python interprete.py`. Escribe `2 * 2` y presiona Enter. Tu programa debe mostrar:

  4.0

- Ejecuta tu programa con `python interprete.py`. Escribe `50 / 5` y presiona Enter. Tu programa debe mostrar:

  10.0

Hora de Comida

Imagina que estás en un país donde es costumbre desayunar entre las 7:00 y las 8:00, almorzar entre las 12:00 y las 13:00, y cenar entre las 18:00 y las 19:00. ¿No sería genial si tuvieras un programa que pudiera decirte qué comer y cuándo?

En `comida.py`, implementa un programa que solicite al usuario una hora y muestre si es `hora de desayuno`, `hora de almuerzo` o `hora de cena`. Si no es hora de una comida, no muestres nada en absoluto. Asume que la entrada del usuario estará formateada en tiempo de 24 horas como `#:##` o `##:##`. Y asume que el rango de tiempo de cada comida es inclusivo. Por ejemplo, ya sea que sean las 7:00, 7:01, 7:59 o 8:00, o cualquier momento intermedio, es hora de desayunar.

Estructura tu programa según lo siguiente, donde `convertir` es una función (que puede ser llamada por `main`) que convierte `tiempo`, un `str` en formato de 24 horas, al número correspondiente de horas como un `float`. Por ejemplo, dado un `tiempo` como `"7:30"` (es decir, 7 horas y 30 minutos), `convertir` debería devolver `7.5` (es decir, 7.5 horas).

```
def main():
    ...

def convertir(tiempo):
    ...

if __name__ == "__main__":
    main()
```

Pistas

- Recuerda que un `str` viene con bastantes métodos, según docs.python.org/3/library/stdtypes.html#string-methods, incluyendo `split`, que separa un `str` en una secuencia de valores, todos los cuales pueden ser asignados a variables de una vez. Por ejemplo, si `tiempo` es un `str` como `"7:30"`, entonces

```
horas, minutos = tiempo.split(":")
```

asignará `"7"` a `horas` y `"30"` a `minutos`.

- Ten en cuenta que hay 60 minutos en 1 hora.

Antes de Comenzar

Ejecuta

mkdir comida

para crear una carpeta llamada `comida` en tu codespace.

Después ejecuta

cd comida

para cambiar de directorio a esa carpeta. Ahora deberías ver el prompt de tu terminal como `comida/ $`. Ahora puedes ejecutar

code hora_comida.py

para crear un archivo llamado `comida.py` donde escribirás tu programa.

Cómo Probar

Así es como probar tu código manualmente:

- Ejecuta tu programa con `python comida.py`. Escribe `7:00` y presiona Enter. Tu programa debería mostrar:

hora de desayuno

- Ejecuta tu programa con `python comida.py`. Escribe `7:30` y presiona Enter. Tu programa debería mostrar:

hora de desayuno

- Ejecuta tu programa con `python comida.py`. Escribe `12:42` y presiona Enter. Tu programa debería mostrar:

hora de almuerzo

- Ejecuta tu programa con `python comida.py`. Escribe `18:32` y presiona Enter. Tu programa debería mostrar:

hora de cena

- Ejecuta tu programa con `python comida.py`. Escribe `11:11` y presiona Enter. Tu programa no debería mostrar nada.

Si estás fallando las verificaciones pero estás seguro de que tu programa se comporta correctamente, asegúrate de que no hayas eliminado la línea `if __name__ == "__main__": main()` de la estructura de código que se te proporcionó. Aprenderás más sobre esto en semanas posteriores.