Creado por:

Isabel Maniega

-1.2- Introducción a Python (Continuación)

-1.2.1- Condiciones Multiples

Operador: |

(condición1) ó (condición2)

In [1]:
```
listado = [5, 10, 15, 20, 25]
listado
```

Out[1]:
```
[5, 10, 15, 20, 25]
```

Buscar números que son o bien 5 o bien 25

In [2]:
```
# Paso a Paso
contador = 0 # inicializar el contador con valor 0

for numero in listado:
    # contador = 0
    if (numero == 5) | (numero == 25):
        contador = contador + 1
        # si el numero es 5 entonces el contador suma 1 --> 0 a 1
        # si el numero es 25 entonces el contador suma 1 --> 1 a 2
    # recorre la lista y finaliza el bucle

print(contador) # Resultado esperado es 2
```

```
2
```

Ahora veremos el uso de print

In [3]:
```
# Paso a Paso
contador = 0 # inicializar el contador con valor 0

for numero in listado:
    # contador = 0
    print('valor del contador antes if: ', contador)
    print('valor del numero: ', numero)
    if (numero == 5) | (numero == 25):
        print('### valor del numero en IF: ', numero)
        contador = contador + 1
        print('*** valor del contador después de IF: ', contador)
        # si el numero es 5 entonces el contador suma 1 --> 0 a 1
        # si el numero es 25 entonces el contador suma 1 --> 1 a 2
    # recorre la lista y finaliza el bucle
```

```
print(contador) # Resultado esperado es 2
```

```
valor del contador antes if:  0
valor del numero:  5
### valor del numero en IF:  5
*** valor del contador después de IF:  1
valor del contador antes if:  1
valor del numero:  10
valor del contador antes if:  1
valor del numero:  15
valor del contador antes if:  1
valor del numero:  20
valor del contador antes if:  1
valor del numero:  25
### valor del numero en IF:  25
*** valor del contador después de IF:  2
2
```

2º opción uso de OR

In [4]:
```
listado = [5, 10, 15, 20, 25]
listado
```

Out[4]:
```
[5, 10, 15, 20, 25]
```

In [5]:
```
# Paso a Paso
contador = 0 # inicializar el contador con valor 0

for numero in listado:
    # contador = 0
    if (numero == 5) or (numero == 25):
        contador += 1
        # si el numero es 5 entonces el contador suma 1 --> 0 a 1
        # si el numero es 25 entonces el contador suma 1 --> 1 a 2
    # recorre la lista y finaliza el bucle

print(contador) # Resultado esperado es 2
```

```
2
```

Operador: &

(condición1) y (condición2)

In [6]:
```
listado = [5, 10, 15, 20, 25]
listado
```

Out[6]:
```
[5, 10, 15, 20, 25]
```

Tenemos una variable además de la lista con valor x = 1, entonces queremos buscar la coincidencia que el valor de x=1 y además que el valor de numero sea 15

In [7]:
```
x = 1
```

In [8]:
```
for numero in listado:
    print('valor del numero: ', numero)
    if (numero == 15) & (x == 1):
```

```
        print('### valor del numero en IF: ', numero)
        print('### valor de x en IF: ', x)
        print("Hemos detectado un 15 en la lista y la x vale 1")
```

```
valor del numero:  5
valor del numero:  10
valor del numero:  15
### valor del numero en IF:  15
### valor de x en IF:  1
Hemos detectado un 15 en la lista y la x vale 1
valor del numero:  20
valor del numero:  25
```

2ª opción: and

In [9]:
```
listado = [5, 10, 15, 20, 25]
listado
```

Out[9]:
```
[5, 10, 15, 20, 25]
```

In [10]:
```
x = 1
```

In [11]:
```
for numero in listado:
    print('valor del numero: ', numero)
    if (numero == 15) and (x == 1):
        print('### valor del numero en IF: ', numero)
        print('### valor de x en IF: ', x)
        print("Hemos detectado un 15 en la lista y la x vale 1")
```

```
valor del numero:  5
valor del numero:  10
valor del numero:  15
### valor del numero en IF:  15
### valor de x en IF:  1
Hemos detectado un 15 en la lista y la x vale 1
valor del numero:  20
valor del numero:  25
```

IN/NOT IN

In [12]:
```
listado = [10, 20, 30]
listado
```

Out[12]:
```
[10, 20, 30]
```

In [13]:
```
# está el valor 10 en la lista?¿ --> Sí == True; No == False
10 in listado
```

Out[13]:
```
True
```

In [14]:
```
10 not in listado
```

Out[14]:
```
False
```

In [15]:
```
20 in listado
```

Out[15]:
```
True
```

In [16]:
```
20 not in listado
```

Out[16]: False

Entrada de texto por teclado

String

In [17]:
```
input("hola como estás?? - Digame: ...")
```

Out[17]: 'Afónica'

Asignación de variable...

In [18]:
```
texto = input("hola como estás?? - Digame: ...")
texto
```

Out[18]: 'Afónica'

Ejemplo de entrada de numeros

In [19]:
```
numero = int(input("dígame su número favorito: <No valen decimales>..."))
numero
```

Out[19]: 2

In [20]:
```
print("Su número favorito es: ", numero)
```

```
Su número favorito es:  2
```

Ejemplo de numeros decimales

In [21]:
```
numero_decimal = float(input("dígame su número decimal favorito: <valen d
numero_decimal
```

Out[21]: 2.3

In [22]:
```
print("Su número decimal favorito es: ", numero_decimal)
```

```
Su número decimal favorito es:  2.3
```

Archivos

Streams

La apertura del stream no solo está asociada con el archivo, sino que también se debe declarar la manera en que se procesará el stream. Esta declaración se llama open mode (modo de apertura).

Si la apertura es exitosa, el programa solo podrá realizar las operaciones que sean consistentes con el modo abierto declarado.

Hay dos operaciones básicas a realizar con el stream:

- **Lectura del stream:** las porciones de los datos se recuperan del archivo y se colocan en un área de memoria administrada por el programa (por ejemplo, una variable).
- **Escritura del stream:** Las porciones de los datos de la memoria (por ejemplo, una variable) se transfieren al archivo.

Hay tres modos básicos utilizados para abrir un stream:

- **Modo Lectura:** un stream abierto en este modo permite solo operaciones de lectura; intentar escribir en la transmisión provocará una excepción (la excepción se llama UnsupportedOperation, la cual hereda el OSError y el ValueError, y proviene del módulo io).
- **Modo Escritura:** un stream abierto en este modo permite solo operaciones de escritura; intentar leer el stream provocará la excepción mencionada anteriormente.
- **Modo Actualizar:** un stream abierto en este modo permite tanto lectura como escritura.

Manejo de Archivos

Nota: nunca se utiliza el constructor para dar vida a estos objetos. La unica forma de obtenerlos es invocar la función llamada open().

La función analiza los argumentos proporcionados y crea automáticamente el objeto requerido.

Si deseas deshacerte del objeto, invoca el método denominado close().

La invocación cortará la conexión con el objeto y el archivo, y eliminará el objeto.

Proceso:

stream = open(file, mode = 'r', encoding = None)

- El nombre de la función (open) habla por si mismo; si la apertura es exitosa, la función devuelve un objeto stream; de lo contrario, se genera una excepción (por ejemplo, FileNotFoundError si el archivo que vas a leer no existe).
- El primer parámetro de la función (file) especifica el nombre del archivo que se asociará al stream.
- El segundo parámetro (mode) especifica el modo de apertura utilizado para el stream; es una cadena llena de una secuencia de caracteres, y cada uno de ellos tiene su propio significado especial (más detalles pronto).
- El tercer parámetro (encoding) especifica el tipo de codificación (por ejemplo, UTF-8 cuando se trabaja con archivos de texto).
- La apertura debe ser la primera operación realizada en el stream.

Nota: el modo y los argumentos de codificación pueden omitirse; en dado caso, se tomarán sus valores predeterminados. El modo de apertura predeterminado es leer en modo de texto, mientras que la codificación predeterminada depende de la plataforma utilizada.

In [23]:
```
# Se abre el archivo tzop.txt en modo lectura, devolviéndolo como un obje
stream = open("tzop.txt", "rt", encoding = "utf-8")

# Se imprime el contenido del archivo:
print(stream.read())
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

In [24]:
```
stream.close()
```

readline()

Si deseas manejar el contenido del archivo como un conjunto de líneas, no como un montón de caracteres, el método readline() te ayudará con eso.

El método intenta leer una línea completa de texto del archivo, y la devuelve como una cadena en caso de éxito. De lo contrario, devuelve una cadena vacía.

Esto abre nuevas oportunidades: ahora también puedes contar líneas fácilmente, no solo caracteres.

In [25]:
```
from os import strerror

try:
    character_counter = line_counter = 0
    stream = open('tzop.txt', 'rt')
    line = stream.readline()
    while line != '':
        line_counter += 1
        for char in line:
            print(char, end='')
            character_counter += 1
        line = stream.readline()
    stream.close()
```

```
    print("\n\nCaracteres en el archivo:", character_counter)
    print("Líneas en el archivo:     ", line_counter)
except IOError as e:
    print("Se produjo un error de E/S:", strerror(e.errno))
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

Caracteres en el archivo: 822
Líneas en el archivo:      19
```

readlines()

Cuando el método readlines(), se invoca sin argumentos, intenta leer todo el contenido del archivo y devuelve una lista de cadenas, un elemento por línea del archivo.

Si no estás seguro de si el tamaño del archivo es lo suficientemente pequeño y no deseas probar el sistema operativo, puedes convencer al método readlines() de leer no más de un número especificado de bytes a la vez (el valor de retorno sigue siendo el mismo, es una lista de una cadena).

Siéntete libre de experimentar con el siguiente código de ejemplo para entender cómo funciona el método readlines():

In [26]:
```
stream = open("tzop.txt")
print(stream.readlines(20))
print(stream.readlines(20))
print(stream.readlines(20))
print(stream.readlines(20))
stream.close()
```

```
['Beautiful is better than ugly.\n']
['Explicit is better than implicit.\n']
['Simple is better than complex.\n']
['Complex is better than complicated.\n']
```

write()

El método se llama write() y espera solo un argumento: una cadena que se transferirá a un archivo abierto (no lo olvides), el modo de apertura debe reflejar la forma en que se

transfieren los datos, escribir en un archivo abierto en modo de lectura no tendrá éxito).

No se agrega carácter de nueva línea al argumento de write(), por lo que debes agregarlo tu mismo si deseas que el archivo se complete con varias líneas.

In [27]:
```
from os import strerror

try:
        file = open('newtext.txt', 'wt') # Un nuevo archivo (newtext.txt)
        for i in range(10):
                s = "línea #" + str(i+1) + "\n"
                for char in s:
                        file.write(char)
        file.close()
except IOError as e:
        print("Se produjo un error de E/S:", strerror(e.errno))
```

El ejemplo en el editor muestra un código muy simple que crea un archivo llamado newtext.txt (nota: el modo de apertura w asegura que el archivo se creará desde cero, incluso si existe y contiene datos) y luego coloca diez líneas en él.

La cadena que se grabará consta de la palabra línea, seguida del número de línea. Hemos decidido escribir el contenido de la cadena carácter por carácter (esto lo hace el bucle interno for) pero no estás obligado a hacerlo de esta manera.

Solo queríamos mostrarte que write() puede operar con caracteres individuales.

Otra forma de leer archivos - with open()

Con esta forma no necesitamos cerrar el stream una vez abierto él lo realizará al finalizar el proceso deseado.

In [28]:
```
with open('tzop.txt') as f:
    contents = f.read()
    print(contents)
```

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

In [29]:
```
#  Actualizar el archivo agregando una línea nueva:

with open("myfile.txt", "a") as f:
    f.write("Today \n")
```

Creado por:

Isabel Maniega