MÓDULO

3

Área: **NEGOCIOS**

Curso:**PROGRAMACIÓN BÁSICA (PYTHON)**

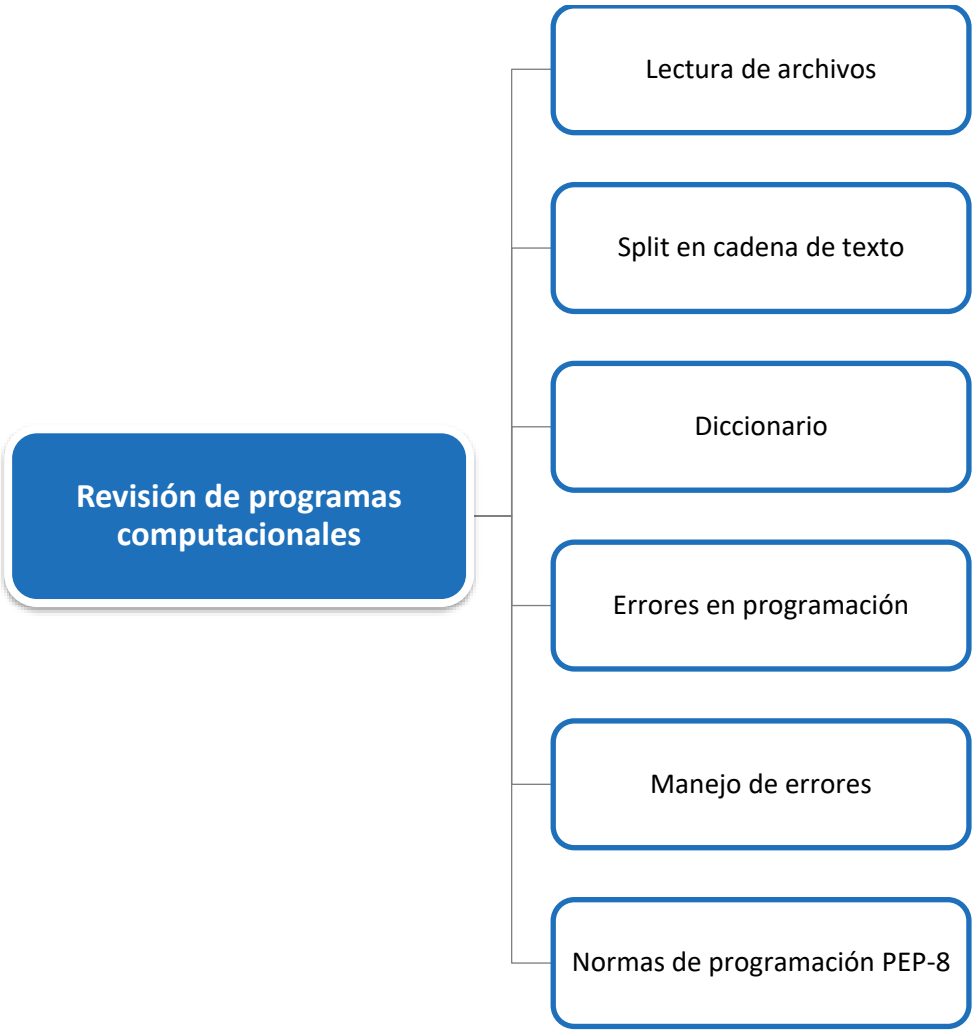
Módulo: Revisión de programas computacionales

IPP

SUEÑA • APRENDE • CRECE

Índice

Mapa de Contenido



RESULTADO DE APRENDIZAJE DEL MÓDULO	Verifica la correcta aplicación de pasos lógicos en la construcción de un programa computacional, para determinar posibles errores de programación.

Introducción

Una situación común en la programación, es la búsqueda de los errores que inducen a fallas graves en la ejecución de un programa desarrollado; a veces esa falla era la falta de un punto y coma (;) por ejemplo. En Python no existen puntos y coma, pero si la **identanción**: en donde un programador o su IDE ocupa 4 espacios en vez de un **tab** y el otro programador ocupa solo tab, lo cual significa que al momento de juntar sus códigos no funciona el programa.

También existían casos (espero que aún no existan) en los cuales los programadores poco éticos, realizan códigos de tal manera que solo es entendible por ellos, y al llegar un nuevo programador no puede entender nada de lo desarrollado por su predecesor. Al realizar estas acciones, los programas se estancan, dado que su actualización solo depende de un programador; por estos motivos se inició la estandarización de la programación, más conocido como “buenas practicas”, como será el **PEP 8**.

En el área de informática se dice lo siguiente:

“Programa que no se actualiza, es el programa que no se utiliza”.

Esto es una realidad. Las necesidades siempre son distintas a través del tiempo; los clientes que ocupan una herramienta constantemente se están actualizando, por seguridad, nuevas aplicaciones, integración con nuevos dispositivos o corrección de errores.

1. Lectura de archivos

En Python3 es muy sencillo saber lo que dice cada archivo, dado que su función **open integrada,** funciona bastante bien. ¿Cómo funciona?

- Crear un archivo, con información .txt
- Guardar el siguiente archivo como archivo.txt

```
linea 1
l2inea 2
palabral 2linea 3
palaba 2linea 4
2linea 5 final
```

Recordar guardar el archivo en la misma carpeta donde está el archivo .py

- Crear un archivo .py

Lectura.py podría ser un buen nombre.

```
archivo = open(“archivo”,”r”)
Print(archivo)
archivo.close()
```

IMPORTANTE

Recomendación: Siempre cierren el archivo, a penas lo dejen de utilizar, eso se recomienda para no gastar recursos innecesarios de su computador, al principio con programas básicos no existirá muchos problemas pero a medida que realicen programas más extensos esto será más notorio, por eso siempre es mejor perfeccionarse desde el inicio y no a mitad de camino

Mostrará lo siguiente:
<_io.TextIOWrapper name='archivo.txt' mode='r' encoding='UTF-8'>

Esto significa que archivo ahora tiene la propiedad de un método de entrada o salida de datos del archivo archivo.txt, que se encuentra en modo r (que significa read, leer en español), y con la codificación UTF-8 (8-bit Unicode Transformation Format), es uno de los formatos más utilizados actualmente en el 2017.

Como también existen diferentes modos para leer un archivo, los cuales son:

modo	Significado	Definición y modo operación
r	Read – leer	Se ocupa solo para leer la información de un archivo
rb	Read bit – leer en binario	Se ocupa solo para leer la información de un archivo en binario
r+	Read plus – leer y escribir	Se ocupa para leer y después escribir en el archivo
rb+	Read bit plus – leer y escribir en binario	Se ocupa para leer y después escribir en el archivo en binario
w	Write – escribir	Escribir en el archivo, si este existe lo reemplaza completamente, si no existe lo crea
wb	Write bit – escribir en binario	Escribir en el archivo en binario, si este existe lo reemplaza completamente, si no existe lo crea
w+	Write plus – escribir y leer	Escribir y leer en el archivo, si este existe lo reemplaza completamente, si no existe lo crea
wb+	Write plus bit– escribir y leer en binario	Escribir y leer en el archivo en binario, si este existe lo reemplaza completamente, si no existe lo crea
a	Add – Añadir	Añade la información después de la última línea del archivo si este no existe lo crea
ab	Add bit- Añadir en binario	Añade la información en binario después de la última línea del archivo si este no existe lo crea
a+	Add plus - Añadir y lectura	Añade y lee la información en después de la última línea del archivo si este no existe lo crea
ab+	Add plus bit - Añadir y lectura binaria	Añade y lee la información en binario después de la última línea del archivo si este no existe lo crea

Si el archivo no existe Python arrojara error. En los archivos con modo de lectura **Read**, dado que estos solo buscan el archivo.

Hay diferentes maneras de recorrer un archivo. La más básica pero no tan recomendada es recorrer cada punto con un **for i in** archivo, si realizamos esto pasara lo siguiente

Código:

```
archivo = open("archivo.txt","r")
for i in archivo:
        print(i)
archivo.close()
```

El resultado en consola será el siguiente:

```
linea 1

linea 2

palabra linea 3

palaba linea 4

linea 5 final
```

¿Por qué tanto espacio?
El motivo es que al final de línea esté un salto de línea programado en el txt, y como el print ya tiene un salto de línea incluido, lo mejor sería borrarlo. Este salto de línea en algunos casos puede ser muy molesto, dado que nos complica en la programación.

Lo mejor es eliminar el salto de línea con **replace**. Cambiamos el replace en la variable **i** por nada, así al momento de mostrar la información se realizara sin salto de línea innecesario.

Entonces el código quedaría de la siente manera:

```
archivo = open("archivo.txt","r")
for i in archivo:
        i = i.replace("\n","")
        print(i)
archivo.close()
```

Mostrando la siguiente información:

```
linea 1
linea 2
palabra linea 3
palaba linea 4
linea 5 final
```

Existen distintos métodos para leer archivo; los programadores son libres de ocupar el que más les acomode. A continuación mostraremos las siguientes maneras de recorrer un archivo:

1.1 Método: readline()

Read line lee una línea del archivo quedando siempre posicionada después en la siguiente. Ejemplo:

Código:

```
archivo = open("archivo.txt","r")
line = archivo.readline()
print(line)
line = archivo.readline()
print(line)
```

Muestra lo siguiente:

```
linea 1

linea 2
```

Se avanza de línea aunque las líneas de comando sean exactamente iguales, eso se debe a que readline siempre queda en el cursor siguiente.

IMPORTANTE

Advertencia: si realizo un readline cuando el curso ya está en la última línea Python retorna un error.

Ahora si queremos recorrer todas las líneas con readline, existe el **readlines**; el cual retorna todas las líneas, esta por lo general se ocupa con un **for**, como por ejemplo:

Código:

```
archivo = open("archivo.txt","r")
for linea in archivo.readlines():
        print(linea)
```

Muestra lo siguiente:

```
linea 1

linea 2

palabra linea 3

palaba linea 4

linea 5 final
```

Ahora es importante saber del **punto de lectura**. El puntero es como una flecha que se mueve a través del documento, en ella es importante saber su posición para saber qué línea mostraremos o dónde escribiremos la nueva información; si no se utiliza bien y estamos trabajando de un modo **r+** y sobrescribiremos información.

Existen comandos para saber la posición del cursor y también moverlo:

seek(byte):	**read(long):**	**tell():**	**write(string):**
•Mueve el cursor a la dirección de byte indicada.	•Lee todo el archivo, si se le entrega un parámetro long, este leera solo hasta el numero de byte que indique long.	•Retorna la posición actual del cursor.	•Escribe en el archivo en la posición actual del cursor, en este caso. Se le entrega por parámetro un string.

Ejercicio:

Realicemos un código para escribir en el archivo sin borrar su información anterior, al final sin utilizar **add** en **modo r+.**

Código:

```
archivo = open("archivo.txt","r+")
contenido = archivo.read() #leemos todo el archivo para que avance el cursor
archivo.write("nuevo texto")
archivo.close()
```

Lo ejecutamos y nos damos cuenta que no muestra nada, dado que no ocupamos ningún print en consola, pero si vamos al archivo .txt veremos lo siguiente:

```
linea 1
linea 2
palabra linea 3
palaba linea 4
linea 5 final
nuevo texto
```

Ahora nos percatamos que se ingresó la nueva línea, al final del documento.

Algunas veces cuando se lee un archivo txt, que contenga acentos el programa Thonny y algunos IDE, nos arrojarán problemas según su configuración inicial, para evitar este problema, agreguen un tercer parámetro al open, que es: encoding='utf-8'

Ejemplo:

```
archivo = open("texto.txt","r", encoding='utf-8')
```

2. Split en cadena de texto

Split es ocupado en la programación para separar un texto en un arreglo con un carácter regular. Por ejemplo se ocupa para separar las líneas en un arreglo o los términos por coma(,).

Con este ejemplo quedara más claro:

Código:

```
texto = 'hola, a todos, esto esta separado'
nueva_arreglo = texto.split(',')
print(nueva_arreglo)
```

Y mostrará lo siguiente:

```
['hola', ' a todos', ' esto esta separado']
```

Este comando nos será muy útil al momento de querer separar información. Split es muy sencillo, split(atributo), atributo es el carácter por el cual quieres separar el texto.

3. Diccionario

Diccionario es una variable que se asemeja bastante a un arreglo, éste puede ir guardando información en ella. La ventaja que tiene a diferencia de tener posiciones de memoria como 0,1,2,3,4,5 hasta n-1, es que el diccionario puede tener una palabra como posición conocida como llave o key. Primero se tiene que declarar, el nombre de nuestro diccionario con llaves.

Ejemplo:

Creemos un arreglo que tenga la variable M Y F, que signifique Masculino y Femenino, y en otro arreglo en forma de diccionario lo iremos contando.

```
arreglo = ['M','M','F','M','M','F','F','M','F','M','F','M','F','F','M','F','F','M','F','F']
sexo={}
sexo["masculino"] = 0
sexo["femenino"] = 0
for i in arreglo:
        if (i == 'M'):
                sexo["masculino"] += 1
        else:
                sexo["femenino"] += 1
print(sexo)
```

El computador mostrará lo siguiente:

```
{'masculino': 9, 'femenino': 11}
```

Ahora realicemos un ejercicio un poco más complicado, con colores, pero ahora no iniciaremos todos los colores en cero dado que no sabemos que colores van a ir saliendo. Si lo hacemos de la misma manera anterior nos arrojará un error, dado que debemos comprobar si existe el diccionario con ese color, para eso existe el comando in, que nos indicará con un True o un False si está en el diccionario.

Ejemplo de código:

```
arreglo =
['rojo','verde','verde','rojo','azul','verde','rojo','azul','azul','amarillo','morado','amarillo','morado','
naranjo','violeta','azul','blanco','azul','negro','negro']
color={}
for i in arreglo:
        if (i in color):
                color[i] += 1
        else:
                color[i] = 1
print(color)
```

Resultado en pantalla:

```
{'rojo': 3, 'verde': 3, 'azul': 5, 'amarillo': 2, 'morado': 2, 'naranjo': 1, 'violeta': 1, 'blanco': 1, 'negro': 2}
```

Que es lo que realiza el código:
Lo primero se inicializa la variable arreglo con todos los colores, posteriormente se inicia la variable color como un diccionario. Ahora comenzamos a recorrer el arreglo dentro de un for, cada carácter tendrá el valor i, entonces ahora en el if buscamos. Si existe el valor que tiene i en el diccionario color, si este existe retornara True y solamente le sumaremos un valor más a nuestro color, pero si este no existe iniciaremos el diccionario del color con el valor de 1. Al final de todo solo imprimiremos el diccionario completo.

También existen otros comandos con los diccionarios, los cuales son:

Keys() :	Values() :	Ítems():
•Este retorna toda las llaves del diccionario.	•Este retorna todo los valores del diccionario.	•Retorna una tupla de cada valor que existe en el diccionario.

3.1 Ejercicio de aplicación

- Realicemos un programa con todo lo aprendido. Creemos una base de datos en un txt y después leamos este archivo y obtengamos información de él de manera útil.

- Somos un Instituto, en donde guardamos toda la información de nuestros alumnos en un txt. Este txt contendrá la siguiente información: rut, nombre, apellido, fecha_nacimiento, carrera, materias (materias estará entre [] y separas con un : cada una)

bd_instituto.txt

```
2121-3, pepito, llamato, 01041983, pedagogia,[introduccion:matematica:lenguaje:sicologia]
1234-3, andres, perez, 01041990, ingenieria,[introduccion:caculo:lineal:programacion]
3242-3, valeria, gonzales, 05041983, contador,[programacion:matematica:comericio:rrhh]
243-3, pamela, ruiz, 23101993, ingenieria,[introduccion:caculo:lineal:programacion]
6754-3, valentina, herrera, 22041990, ingenieria,[estructura:caculo2:ecuaciones:aplicaciones]
5433-3, sebastian, rojas, 27071983, pedagogia,[lenguaje2:matematica:ciencias:sicologia]
543543-3, persona1, apellido1, 12091983, contador,[programacion:matematica:comericio:rrhh]
```

*Lo que se encuentra en naranjo son comentarios.

1. **Guardar la información de cada persona en un string con llave que sea su rut.**

```
archivo = open("bd_instituto.txt","r+")
persona = {}
aux = [] # creamos un arreglo auxiliar es el que contendrá la información dentro del ciclo
for linea in archivo:
        linea = linea.replace('\n','') # borramos el salto de línea
        aux = linea.split(',')# separamos la línea en aux para obtener la información de persona separada
        persona[aux[0]] = aux[1:]
print(persona)
```

Resultado en pantalla:

```
{'2121-3': [' pepito', ' llamato', ' 01041983', ' pedagogia', '[introduccion:matematica:lenguaje:sicologia]'], '1234-3': [' andres', ' perez', ' 01041990', ' ingenieria', '[introduccion:caculo:lineal:programacion]'], '3242-3': [' valeria', ' gonzales', ' 05041983', ' contador', '[programacion:matematica:comericio:rrhh]'], '243-3': [' pamela', ' ruiz', ' 23101993', ' ingenieria', '[introduccion:caculo:lineal:programacion]'], '6754-3': [' valentina', ' herrera', ' 22041990', ' ingenieria', '[estructura:caculo2:ecuaciones:aplicaciones]'], '5433-3': [' sebastian', ' rojas', ' 27071983', ' pedagogia', '[lenguaje2:matematica:ciencias:sicologia]'], '543543-3': [' persona1', ' apellido1', ' 12091983', ' contador', '[programacion:matematica:comericio:rrhh]']}
```

Recuerden que es un diccionario y no se muestra siempre de una manera muy amigable.

2. **Mostrar las materias y el número que esta se repite.**

```
archivo = open("bd_instituto.txt","r+")
persona = {}
aux = [] # creamos un arreglo auxiliar es el que contendrá la información dentro del ciclo
for linea in archivo:
        linea = linea.replace('\n','') # borramos el salto de línea
        aux = linea.split(',')# separamos la línea en aux para obtener la información de persona separada
        persona[aux[0]] = aux[1:]
#ahora tenemos toda la información en persona
materias = {} #creamos el diccionario de materias
for per in persona:
        '''
        como per solo tiene el rut de cada persona, lo que realizamos es obtener el arreglo completo
        guardado en cada persona, que contiene lo siguiente en las siguientes posiciones:
        0 = nombre
        1 = apellido
        2 = fecha_nacimiento
```

```
        3 = carrera
        4 = las materias
        Ahora necesitamos las materias, entonces ocuparemos la posición 4.
        esto se puede hacer de 2 maneras.
        1- obtener el arreglo en una variable auxiliar y después obtener la posición
        aux = persona[per]
        s_materia = aux[4]

        2- obtener la posicion del arreglo del manera directa
        s_materia = persona[per][4]
        '''
        s_materia = persona[per][4] #ahora s_materia tiene la información de las materias
        '''
        s_materia contiene ahora un string con la siguiente estructura
        [introduccion:matematica:lenguaje:sicologia]

        lo primero que realizaremos es eliminar los corchetes y después separaremos por :
        '''
        s_materia = s_materia.replace('[','')
        s_materia = s_materia.replace(']','')
        s_materia = s_materia.split(':') # ahora s_materia tiene un arreglo con cada materias
        for materia in s_materia: #vemos si existe la key se suma si no se crea
                if materia in materias:
                        materias[materia] +=1
                else:
                        materias[materia] = 1
print(materias)
```

Como verán este ejercicio fue algo complicado hubo que utilizar for dentro de for para poder obtener la información.

Resultado en pantalla:

```
{'introduccion': 3, 'matematica': 4, 'lenguaje': 1, 'sicologia': 2, 'caculo': 2, 'lineal': 2, 'programacion': 4, 'comericio': 2, 'rrhh': 2, 'estructura': 1, 'caculo2': 1, 'ecuaciones': 1, 'aplicaciones': 1, 'lenguaje2': 1, 'ciencias': 1}
```

3. Mostrar las carrera y ver cuánto se repiten

```
archivo = open("bd_instituto.txt","r+")
persona = {}
aux = [] # creamos un arreglo auxiliar es el que contendrá la información dentro del ciclo
for linea in archivo:
        linea = linea.replace('\n','') # borramos el salto de línea
        aux = linea.split(',')# separamos la línea en aux para obtener la información de persona separada
        persona[aux[0]] = aux[1:]
#ahora tenemos toda la información en persona
carreras = {}
for per in persona:
        s_carrera = persona[per][3]
        if s_carrera in carreras:
                carreras[s_carrera] += 1
        else:
                carreras[s_carrera] = 0
print(carreras)
```

Resultado en pantalla:

```
{' pedagogia': 1, ' ingenieria': 2, ' contador': 1}
```

4. Errores más comunes en la programación

4.1 El uso de variables inexistentes en el código

Muchas veces ocurre que estamos programando y pensamos que tenemos una variable que siempre declaramos y la ocupamos, al momento de ejecutar el código nos arroja un error, ejemplo:

```
suma = a + 14
Print(suma)
```

La variable **a** nunca existió entonces el computado entregó lo siguiente:

```
Traceback (most recent call last):
  File "archivo.py", line 1, in <module>
    suma = a + 14
NameError: name 'a' is not defined
```

Siempre Python entrega la línea del error: en este caso es la 1; mostrando cuál es. Posteriormente en el **NameError** nos entrega una pequeña reseña de porque falló su ejecución, en este caso dice que la variable **a** no está definida.

4.2 Tipografía

Muchas veces ocurre que al momento escribir rápido se invierte el nombre de una variable o esta queda mal escrita. Ejemplo:

```
Layers = 15
Print(Layesr)
```

El error que entrega es que la variable no existe

4.3 El uso de mayúsculas y minúsculas

Algunas veces ocupamos las variables con mayúsculas o minúsculas, pero ocuparlas indistintamente nos puede costar errores ya que Python busca la palabra exacta, con las mismas mayúsculas y minúsculas. El error que entrega es que la variable no existe.

4.4 Tabulación o espacios

Esto es muy importante dado que si tenemos una función, validación o ciclo, se presentará con un espacio; en Python es muy importante ya que identifica si la acción está dentro o fuera de algún ciclo. Muchas veces ocurre que los programadores programan en algún lugar o copian un código de algún lado y al momento de pegarlos en sus códigos estos vienen con espacios en vez de tab.

Ejemplo:

```
palabra = "hola"
for i in palabra:
        print("la letra es: ") #esta con Tab
       print(i) #esta con espacios
```

Nos mostrara el siguiente error:

```
  File "archivo.py", line 4
    print(i)#esta con espacios
                          ^
IndentationError: unindent does not match any outer indentation level
```

Nos informa que no está en el mismo nivel de indentacion.

4.5 Dos puntos (:) al final de cada ciclo o validación

Cuando realizamos un ciclo while, for o un if, siempre al final de esto deben ir dos puntos (:), los cuales indiquen que después de ellos estarán las instrucción. Ya que en Python no son muy utilizados los **:** por lo general los programadores lo omiten.

```
palabra = "hola"
for i in palabra
        print(i)
```

El error que nos dará será:

```
  File "archivo.py", line 2
    for i in palabra
                 ^
SyntaxError: invalid syntax
```

Nos indicará que la sintaxis ocupada es invalida.

4.6 Variables que empiecen con _ ,- , numero o carácter

En Python no se pueden utilizar variables con _ (guión bajo), – (guión), numero o caracteres extraños (%$*# etc.)

4.7 Guardar archivo sin extensión .py

Aunque en Linux o OSX no existe diferencia entre un documento sin extensión o con extensión .py, en Windows si el archivo no tiene la extensión nos entregará error de documento incompatible.

4.8 Ocupar tipo de dato equivocado

Esto ocurre cuando se quiere ocupar un número y juntarlo con un string o viceversa.

Ejemplo:

```
a = 15
palabra = "el número a es: " + a
print(palabra)
```

Nos entregará el siguiente error:

```
Traceback (most recent call last):
  File "archivo.py", line 3, in <module>
    palabra = "el numero a es: " + a
TypeError: must be str, not int
```

Nos explica que se tiene que trabajar con string, no con una variable número. Para arreglar este error hay que castear la variable a que sea string con str().

5. Manejo de errores

Existe una forma muy sencilla que el try para el manejo de los errores. Trabaja de una forma muy parecida a un if, si todo está bien el código estará en try, pero si existe algún problema el código estará en el except.

Ejemplo:

Hagamos un ciclo infinito que reciba todo el tiempo un número, este lo vamos cambiando a entero. Todo estará dentro del try, entonces como ingresamos el número, sabemos que si ingresamos una palabra o letra esta no se podrá convertir a entero y deberá arrojar un error, entonces en ese momento deberá saltar al except.

Código:

```
while True:
        try:
                x = int(input("ingrese un número: "))
        except ValueError:
                print("no se ingreso un caracter correcto")
                break
```

Entonces en la consola realizaremos lo siguiente:

```
ingrese un número: 32
ingrese un número: 432
ingrese un número: 34
ingrese un número: 23342
ingrese un número: a
no se ingreso un caracter correcto
```

Esta sería una manera de ocuparlo, aunque también lo podemos hacer para forzarlo a que entregue un carácter correcto.

Código:

```
while True:
        try:
                x = int(input("ingrese un número: "))
                break
        except ValueError:
                print("ocurrio un error en el numero ingresado favor intentar de nuevo")
```

Entonces en la consola realizaremos lo siguiente:

```
ingrese un número: sa
ocurrio un error en el numero ingresado favor intentar de nuevo
ingrese un número: ds
ocurrio un error en el numero ingresado favor intentar de nuevo
ingrese un número: a
ocurrio un error en el numero ingresado favor intentar de nuevo
ingrese un número: 12
```

De esta manera trabaja el try. Se puede ocupar en todo el código, no es necesario que sea en ciclos. Sin el try, al momento que ocurra un error, el programa se pararía; con su uso evitamos y controlamos esta situación.

6. Normas de programación PEP-8

En el mundo de la programación, rara vez se ve que un código complejo lo realice solo una persona, por el contrario siempre son varias. ¿Te imaginas qué pasaría si cada una de ellas programara de la manera que quisiera? ¿Qué ocurriría cuando le muestren su código o a otra persona? O bien si el programador está implementando un proyecto y se enferma, luego su reemplazo se percata que el programador tiene una manera muy desordenada de programar y se demora semanas solo en entender cómo programa el programador oficial.

Antiguamente era así: el mundo de la programación era muy desordenado y cada uno programaba con su estilo. Esto ha ido cambiando con el tiempo, dado que se crearon "buenas costumbres de programación": las cuales determinan la manera de declarar una variable, función y hasta los comentarios. En Python se realizó un estándar de programación que es llamado PEP – 8.

6.1 Que es PEP 8

PEP, es el acrónimo de las palabras en inglés "Python Enhancement Proposals", que en español es "**Propuestas de Mejora Python**". Consiste en una pequeña guía realizada para mejorar las prácticas en la programación de Python, fueron creadas el 2001 por Guido van Rossum, Barry Warsaw y Nick Coghlan. Están disponibles en su propia página web: https://www.python.org/dev/peps/pep-0008/

6.1 Propuestas de Mejora de PEP 8

a. Indentación

Nunca mezclar tabulaciones con espacios y si se ocupa espacios que estos siempre sean 4. A excepción que sea un código antiguo que no queramos arruinar, ahí se pueden utilizar 8 espacios por tabulación.

b. Máximo de 79 caracteres por línea

Si estás haciendo un código y este es muy largo se puede agregar un \ al final para continuar en la siguiente línea, aunque también se recomienda encerrar la operación en paréntesis (), y en la siguiente línea cerrar este.

c. Argumentos de funciones ordenados

Al momento de declarar funciones excedes de los 79 caracteres, se deben colocar los argumentos abajo de los suyos o de manera alineada.

Ejemplo:

```
Correcto:
#opcion1
variable = funcionx(variable_1, variable2
                    variable_3)
#opcion2
variable = funcionx(
           variable_1, variable2
           variable_3)
Incorrecto:
variable = funcionx(variable_1, variable2
           variable_3)
```

Se percatarán que si no están todas las variables abajo del nombre de la función, no está ninguna.

d. Validación de un Booleano

Nunca se tiene que validar un booleano con un ==. Ejemplo la variable **valido** es de tipo booleano:

```
Correcto:
if valido:
   Accion1

Incorrecto:
if valido == True:
   accion1
```

e. Espacios

Nunca usar espacios innecesarios en el código. Nunca se tiene que ocupar espacios en operaciones aritméticas:

```
Correcto:
   Resultado = a+b
Incorrecto:
   Resultado = a + b
```

Si existe alguna operación matemática que tenga una complejidad o una prioridad distinta ahí se pueden usar pero no abusar de esto. Ejemplo:

```
Correcto:
c = (a+b) * (a-b)
Incorrecto:
c = (a + b) * (a - b)
```

Nunca usar espacios alrededor de un signo igual en una entrega de argumentos por defecto. Ejemplo:

```
Correcto:
def suma(a=0, b=0):
Incorrecto:
def suma(a = 0, b = 0):
```

No usar espacios blancos incensarios en función:

```
Correcto:
funcion1(2)
Incorrecto:
funcion1 (2)
```

f. Comentarios

No se deben realizar comentarios obvios en el código. Los comentarios no deben contradecir el código.

g. Importación de librerías

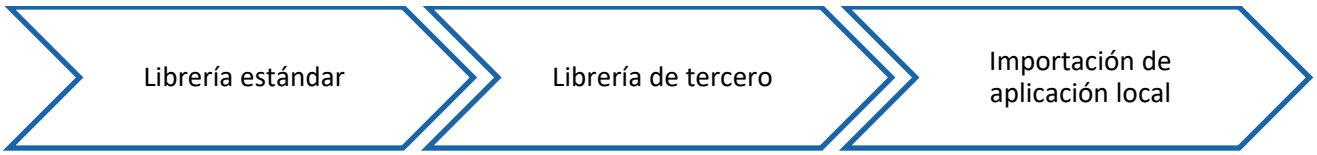
Al momento de importar librerías, éstas no deben ir juntas si no separadas:

```
Correcto:
import os
import sys

Incorrecto:

import os, sys
```

Se permitirá el uso de importación consecutivo cuando vengan solo de la misma librería. Ejemplo:

```
from urllib2 import urlopen, Request
```

Las importaciones siempre tienen que estar al inicio del código y con el siguiente orden.



7. Cierre

Los contenidos vistos en este módulo son fundamentales para la formación de un programador, ya que conocieron el manejo de archivos con Python, cuáles eran sus modos y la importancia de cada uno. Con esta información el estudiante puede comenzar a crear programas con una gran cantidad de caso de pruebas, diccionario y Split. También les será de gran ayuda al momento de la programación, dado que estas son las herramientas más utilizadas en Python.

Posteriormente comprendieron la conducta de los programadores, revisando algunos de los errores más comunes que ocurren durante la programación. Finalmente el conocimiento de la norma de PEP-8 permitirá pulir su forma de programación, realizando una programación más universal, esto les servirá para su desarrollo profesional, ya que sus códigos ahora podrán ser entendibles y perdurables en el tiempo, ser por ustedes o por otros programadores.