MÓDULO

2

Área: **NEGOCIOS**

Curso:**PROGRAMACIÓN BÁSICA (PYTHON)**

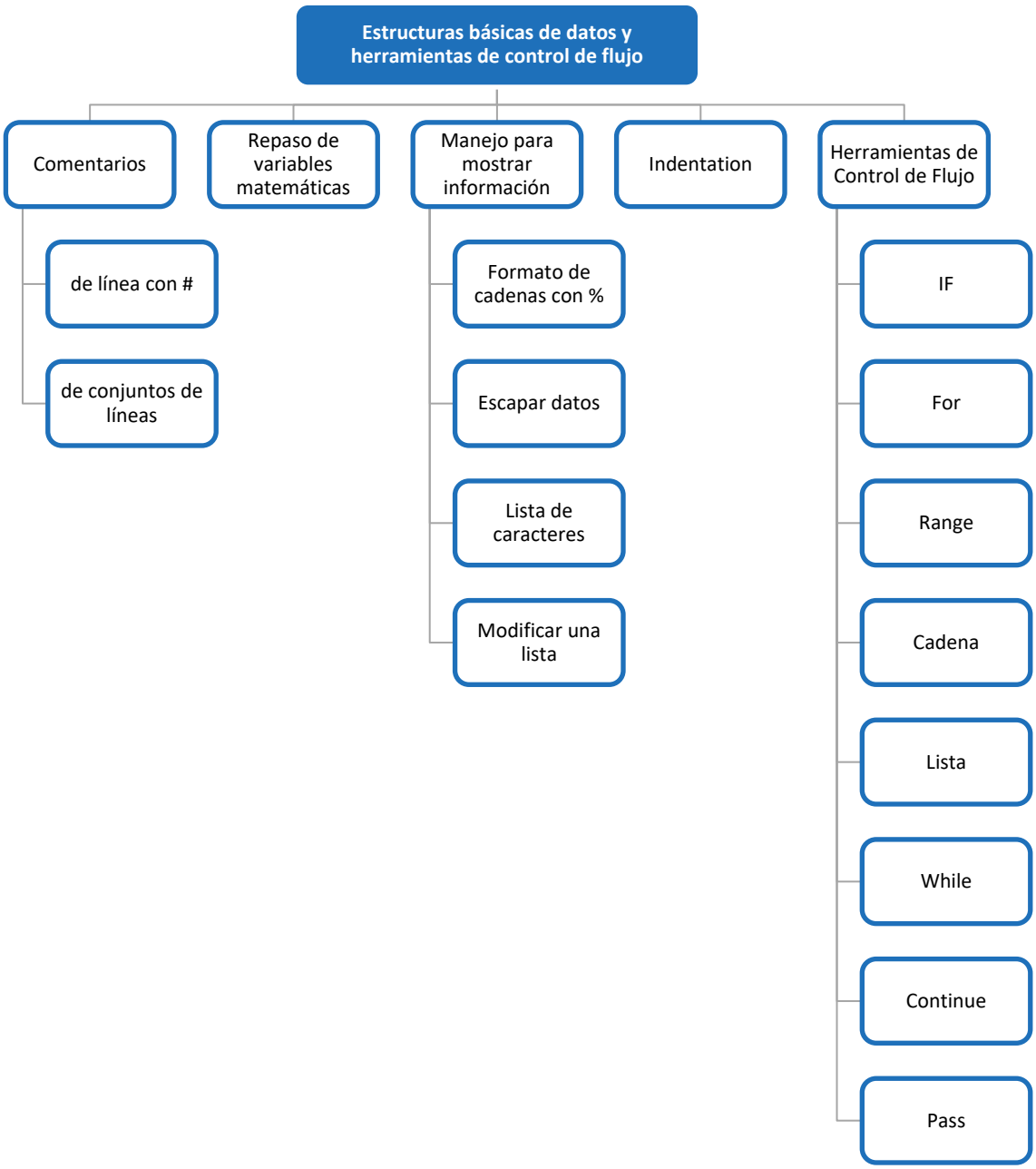
Módulo: Estructuras básicas de datos y herramientas de control de flujo

IPP

SUEÑA • APRENDE • CRECE

Índice

Mapa de Contenido



Resultado de aprendizaje del módulo	• **Crea programas simples a partir de estructuras de datos y herramientas de control de flujo, que responden a enunciados escritos.**

Introducción

Una de las operaciones más importante en la computación que son las validaciones y los ciclos, en los cuales un código podrá tomar una dirección diferente dependiendo de la condición que se le asigne; como por ejemplo escribir **niño**, si el número es menor a 18 o **adulto** si es lo contrario.

Igual de importante que las condiciones son los ciclos. Estos serán utilizados cuando existan acciones que se repitan constantemente y no tenga variaciones. Los ciclos pueden ser infinitos, por una cantidad determinada o por una condicionante.

1. Comentarios

A medida que vamos avanzando en Python, nuestros códigos son cada vez más largos y complejos. Para ello es muy importante saber entregar la información correcta para que la siguiente persona que ocupe o vea tu programa lo entienda con claridad; para esto existen los comentarios en Python.

IMPORTANTE

Los comentarios son línea de códigos que el interpretador de Python omite por completo y así se pueden explicar qué tipo de procesos se realizan o para que sirve cada operación.

* Advertencia: en Python los comentarios no pueden llevar tilde

En Python existen 2 tipos de comentarios, los comentarios que son por línea y los que son por un conjunto de líneas.

1.1 Comentarios de línea con

Ejemplo:

```
A = 5
B = 7
# C es la variable que realizara la suma de las variables a y b
C = a+b  # aca es la variable que realiza la suma
```

El # le dice al código que todo lo que se encuentra a la derecha de él es un comentario.

1.2 Comentarios de conjunto de líneas

Se inicia con “”” (3 comillas dobles) y se termina con “”” (3 comillas dobles).

Esto sirve para realizar explicaciones largas y por lo general acá van la firma del autor y sus explicaciones.

Ejemplo:

```
"""
En este programa se realizaran la suma de 2 variables una de ellas sera la variable a que contendra un numero entero y la variable b que contendra un numero decimal.
Para la realizacion de una suma correcta es necesario. Que los valores asignados tengan ese tipo de valores.
Programa creado el julio del 2017, con finalidad de enseñar a sumar

"""
a = 5
b = 2.3
c = a + b
print(c)
```

2. Repaso de variables matemáticas

Ya conocimos anteriormente las siguientes funciones matemáticas en Python.

Sumar (+) | Restar (-) | Multiplicar (*) | Dividir (/) | Resto (%)

A estas operaciones se le agrega una que tendrá gran utilidad: calcular potencias. Anteriormente y en otros lenguajes de programación, se tenían que agregar otras librerías matemáticas para poder realizar esta operación. pero Python3 viene con una operación ya incorporada que puede realizar potencias.

Potencias (**)
(doble asterisco)

Ejemplo:

```
Variable = 3**2
#el resultado de variables será 9 dado que 3^2 es igual a 9
```

Nos faltan las raíces, pero recuerden que una **raíz es igual que una potencia fraccionaria**, es decir que $\sqrt{4}$ = $4^{1/2}$, entonces para términos de programación:

```
raíz = 4**(1/2)
# el resultado debería ser 2.
```

Esto también ocurre para las raíces enésimas.

3. Manejo para mostrar información

Anteriormente, hemos trabajado con print para mostrar una variable o un número por separado, o si queremos mostrar una información más elaborada generábamos un variable con la información, como por ejemplo:

```
A = 5
Texto = “la variable a es:” + A
Print(Texto)
```

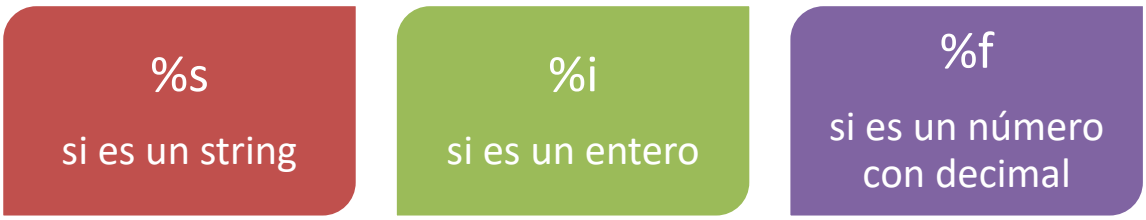
Esto se mostrará por pantalla:

```
La variable a es:5
```

Lo anterior es un hecho muy práctico, pero cuando ya tenemos un gran manejo de datos y queremos optimizar la memoria del computador lo mayor posible, existe otro método de mostrar los mensajes que es método del %.

3.1 Formato de cadenas con %

En este formato sirve principalmente para explicarle al print donde irá el mensaje y de qué tipo será:



Si se quiere mostrar solo una cierta cantidad de decimales se escribe %.3f, en este caso solo mostrara 3 decimales) y posterior a declarar todas las variables se agrega un % y después las variables a utilizar.

Ejemplo:

```
A = 5
B = 2.3
C = A + B
Text = “aca esta un lindo texto”

print (“la información mostrada de la operación es %i + %.1f = %.1f, colocando después un %s”%(A,B,C,Text))
```

La respuesta que mostrará el computador será la siguiente:

```
la información mostrada de la operación es 5 + 2.3 = 7.3, colocando despues un aca esta un lindo texto
```

Ejemplo:

```
Nombre = “Sebastian”
print (“mi nombre es: %s”%Nombre)

print(“el %s blanco de %s, no era %s”%(“caballo”,”napoleon”,”blanco”))
```

Además de estos ejemplos, hay millones de formas de ocupar el print con el %, dadon mayor rapidez al mostrar los mensajes; a diferencia de la forma anterior en que teníamos que castear las variables a string para poder unirlas en una oración.

3.2 Escapar datos

Escapar datos, hace referencia a las veces en que quiero mostrar por pantalla un carácter que Python lo tiene ocupado en otra labor.

Un ejemplo fácil de reconocer es el “. Ya que si escribo en el print un “, Python piensa que es el inicio o el final de un string. Para poder mostrar estos datos solo se necesita agregar un \ antes del carácter, como por ejemplo:

```
print(“ ahora se mostrara una \“ comida doble en la consola de python”)
```

3.3 Lista de caracteres

Pensemos que en una sola variable queremos guardar varios datos que tiene relación con lo mismo, como por ejemplo las notas de un alumno.

N1 = 4.0
N2 = 3.2
N3 = 5.0
N4 = 6.0

Pero si se dan cuenta esto tiene relación con lo mismo, por este motivo es que se crearon las listas.

Notas = (4.0, 3.2, 5, 6)

Los arreglos en Python son leídos desde 0 hasta n-1, siendo n el largo del arreglo. En nuestro ejemplo la primera nota seria la posición [0] con un 4.0 y la última posición seria [3] con un 6.0.

```
---+---+---+---+---+---+-
 P | Y | T | H | O | N |
---+---+---+---+---+---+-
 0   1   2   3   4   5
```

Mostrando como funciona una lista de datos, tenemos la palabra Python la cual está dividida por caracteres.

IMPORTANTE

Nota: Las lista de datos se pueden escribir tanto con () o con [] (La mayoría de la literatura se escribe con [], por lo menos en lo que llevamos de materia sería lo mismo.)

Al realizar las con (), son mayormente conocidas como Tuplas, pero estas solo se pueden definir una vez y no modificarte, a diferencias de las declaradas con [] que son conocidas como listas, estas si se pueden modificar con facilidad solo igualando la posición a la nueva características.

Palabra = (“P”,“Y”,“T”,“H”,“O”,“N”) # sería lo mismo que escribir Palabra = [“P”,“Y”,“T”,“H”,“O”,“N”]

Entonces si escribo esto:

```
print(Palabra[0])
#lo que el programa deberá mostrar seria P
print(Palabra[3])
#lo que el programa deberá mostrar seria H
```

Así es con todos las listas, independiente de lo que contengan, siempre será de 0 hasta n-1, SIEMPRE!, también como buena costumbre genere lista siempre con el mismo tipo de datos, si son notas solo colocar números con decimales no con palabras, dado que si en la lista se agrega una palabra, toda la lista pasa automáticamente a ser string.

Las listas tienen sus propias funciones que nos facilitan mucho la vida. Las funciones son:

len(): Con esta función uno puede obtener el largo de la lista

Ejemplo:

```
Largo = len(Palabra)
#Largo ahora contendrá una variable de tipo entero con el valor de 6.
```

Recuerden que solo se pueden obtener el máximo valor n-6.

Entonces si por algún motivo escribimos:

```
Palabra[6]
```

Python nos arrojara un error diciendo que el índice de la lista está fuera de rango.

Ahora veremos cómo mostrar información:

Ya conocemos como mostrar un carácter. Simplemente encerrando entre corchete, la posición que queremos mostrar, pero ¿qué ocurre si queremos mostrar más de una posición? Para esto existen los rangos, los cuales se refieren [desde donde: menor que esta posicion]

Siguiendo con el ejemplo Palabra, esto sería:

```
Palabra[1:2]

El computador solo mostrará “Y”
```

Esto ocurre porque le dijimos que muestre desde la posición 1 hasta la posición menor a 2, entonces solo el único número que cumple esa condiciones 1.

Ahora si escribimos el siguiente comando:

```
Palabra[1:3]

El computador mostrará [“Y”,”T”]
```

Por el mismo concepto del anterior que número empieza en 1 y es menor que 3, las únicas posiciones que cumplen esta condición son 1 y 2 solamente.

Ahora digamos que quiero mostrar los primero 4 datos, esto se tendría que hacer.

```
Palabra[:4]

El computador nos mostrará [“P”,”Y”,”T”,”H”]
```

Esto ocurre porque le dijimos al computador que nos obtenga datos desde el inicio hasta el número menor a 4, es decir 0, 1, 2 y 3 que fueron lo que se desplegaron.

Ahora si quisiéramos mostrar los últimos 2 datos de la lista se realizaría de la siguiente manera.

```
Palabra[4:6]
```

El computador nos mostrará [“O”,”N”]

O también puede ser Palabra[4:]

Con estas 2 sentencias se obtiene el mismo resultado.

3.4 Cómo modificar una lista

IMPORTANTE

Nota: Los comandos que se verán a partir, solo funcionarán si se declararon las lista con []. Debido a que con () Python lo interpreta como Tuplas de datos, los cuales solo podemos realzar lo enseñado anteriormente.

3.4.1 Agregar datos a una lista

i. Append

Para agregar datos a una lista se realiza con el comando append(valor). Ejemplo:

Palabra.append(3)
Y ahora palabra quedaria como ['P', 'Y', 'T', 'H', 'O', 'N', 3]

Palabra.append(“S”)
Y quedaria ['P', 'Y', 'T', 'H', 'O', 'N', 3, ’S']

Siempre append agrega la variable al final de la lista.

ii. Insert

Si quisieramos insertar un dato pero entremedio de la lista, esto se realiza con el comando insert(posicion, valor). Ejemplo: ahora separemos el 3 de la S en palabra.

Palabra.insert(7,” ”)
Y quedaría ['P', 'Y', 'T', 'H', 'O', 'N', 3, ' ', ’S']

Y si ahora queremos separar el 3, seria
Palabra.insert(6,” ”)
Y quedaría ['P', 'Y', 'T', 'H', 'O', 'N', ' ', 3, ' ', ’S']

Y ahora queremos agregar una E antes de la S, entonces seria
Palabra.insert(9,”E”)
Quedando ['P', 'Y', 'T', 'H', 'O', 'N', ' ', 3, ' ', 'E', ’S']

iii. Remove

Si quisiéramos eliminar un dato que está en nuestra lista, esto se hace con el comando remove(valor)

Digamos que ahora queremos eliminar los espacios de nuestro arreglo:

Palabra.remove(“ ”)
Quedando ['P', 'Y', 'T', 'H', 'O', 'N', 3, ' ', 'E', ’S']

IMPORTANTE

Nota: Se dieron cuenta que quedo igual un “ ”(espacio), esto es debido a que remove busca la primera coincidencia de izquierda a derecha una vez que la encuentra la borra y no sigue buscando.

Advertencia: al borrar un dato que no existen en el arreglo Python arrojara un error y dejará de ejecutar todo el código.

iv. Búsqueda

Para buscar un valor dentro de mi lista, bastará solo con poner el valor a buscar “in” y después el arreglo. Siguiendo con el mismo ejemplo:

“Y” in Palabra
El resultado que entrega Python es True.

Pero si buscamos:

82 in Palabra

El resultado que entrega Python es False.

v. Obtener índice de una búsqueda

Si queremos buscar un objeto en una lista, pero en vez que nos responda True o False, si no su posición de donde está el archivo buscado.

Ejemplo:

Palabra.index(“T”)
El resultado será 2.

IMPORTANTE

Nota: Recuerden que los índices son de 0 hasta (n-1).

Advertencia: Si se busca una variable que no existe en la lista, con index y este no existe, Python entregara un error y se dejara de ejecutarse.

Si estamos buscando una variable que existe 2 veces entre una lista.

Lista = [23,5,3,23,5,7]
Y buscamos el 5, entregara el primer valor de izquierda a derecha.
El resultado seria 1.

vi. Cadena de Texto (String)

Cuando creamos una variable de cadena de caracteres string, esta es automáticamente una Lista de caracteres (Char).

IMPORTANTE

Nota: Una cadena de texto se comporta como una Dupla, teniendo comando específicos para su modificación.

Por este motivo se pueden ocupar las mismas funciones que se vieron en las listas.

```
Variable = “Hola Mundo IPP”
Variable[2] el resultado sera l
Variable.index(“a") el resultado seria 3
len(Variable) el resultado seria 14
```

Pero String, tiene otras nuevas funciones que son propias de las cadenas de texto.

lower(), que cambia todo el texto a minúsculas.
upper(), que cambia todo el texto a mayúsculas.

```
print( Variable.lower() ) la respuesta de Python será: hola mundo ipp
print( Variable.upper() ) la respuesta de python sera: HOLA MUNDO IPP
```

También las cadenas de caracteres se pueden multiplicar en Python3.

Ejemplo:

```
New_variable = 3*” Programas ”
El resultado será:
' Programas  Programas  Programas ‘
```

También se pueden reemplazar algunos valores. Con el comando replace(valor_antiguo, valor_nuevo)

```
New_variable.replace(“Programas”, “Cambiamos la variable”)

El resultado sería:
' Cambiamos la variable  Cambiamos la variable  Cambiamos la variable ‘
```

ACTIVIDAD

Ejercicios formativos:

Crear un pequeño texto, no más de 5 palabras.

1. Mostrar por pantalla los primeros 6 caracteres del texto creado.
2. Mostrar por pantalla los últimos 4 caracteres del texto creado.
3. Mostrar por pantalla entre tercer y décimo carácter del texto creado.
4. Al texto creado reemplace la letra “a" con “ipp”
5. Busque en la cadena de caracteres la posición de la primera i.
6. Cree una lista de datos, que contenta string y números.
7. Agrega una variable en el texto, en la 4 posición.
8. Elimine la información de la posición 3.

4. Indentation

Múltiples lenguajes de programación, tienen su sintaxis correspondiente para poder distinguir si una acción está dentro y fuera de un ciclo o acción, en algunos lenguajes son llaves, ; o paréntesis, pero en Python es la indentacion.

¿Qué es la Indentation?

Un código bien identado es cuando realizo un función o acción y todo lo que esta acción contenga se encuentra un tab(->) hacia la derecha.

Ejemplo:

```
Def mi_funcion():
        a = 15
        print(“imprimo este mensaje”)
print(“Estoy fuera de la función”)
```

En este ejemplo se ve que la variable a y el print de ¨ imprimo este mensaje ¨, se encuentra tabulados, entonces ellos, pertenece dentro de la función.

En Python se puede identear de 2 maneras, con un TAB o con 4 ESPACIOS.

IMPORTANTE

Advertencia: Si en un código Python estas trabajando con TAB NO SE PUEDE CAMBIAR A 4 ESPACIOS, o al revés si se trabaja con 4 ESPACIO NO SE PUEDE CAMBIAR A TAB. Es por un simple hecho que el interpretador no sabrá distinto que pertenece a una función o acción.

5. Herramientas de control de Flujo

IMPORTANTE

Recomendación: Se les recomienda trabajar todo este módulo en archivo .py, y solo ejecutarlos, dado que van a tener que están escribiendo varias líneas de código para que se pueda mostrar algún resultado y escribiendo directo en la consola de Python sería un poco complicado.

Todo lo que hemos visto hasta el momento, es cómo el programa trabaja linealmente, es decir solo avanza. Luego de ello, haremos un programa de manera más inteligente, es decir: que pueda ahora tomar decisiones, que repita un flujo *n* cantidad de veces o que ejecute algo hasta que se cumpla una condición. Para ello existen distintas herramientas de flujo:

5.1 IF

IF, en Python es la representación de SI, esta herramienta es muy útil dado que sirve para ver si se cumple una condición ocurre algo y si no realice otra acción o nada.

Después de cada if tiene que terminar la línea en : (dos puntos), y después colocar el salto de línea y identar.

Ejemplo:

```
Edad = input(“Ingrese su edad”)
Edad = int(Edad)
If (Edad >= 18):
        print(“ Es mayor de edad”)
```

Al ejecutar estas líneas de código, le estamos diciendo que vea la edad que le asignamos y si esta es mayor a o igual a 18, muestre el mensaje, y si no cumple esa condición no realice nada.

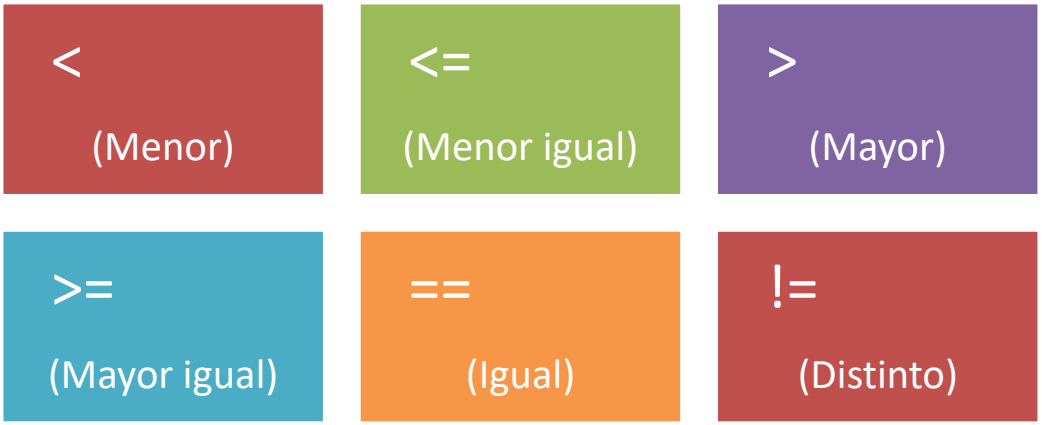
Pero esto no es muy útil, dado que nosotros queremos que muestre algo cuando se cumple o si no se cumple. Para esto existe un co-comando que es **ELSE**, es la representación de SINO. Siguiendo el mismo ejemplo la estructura seria la siguiente:

```
Edad = input("Ingrese su edad")
Edad = int(Edad)
if (Edad >= 18):
        print(" Es mayor de edad")
else:
        print("Es menor de edad")
```

IMPORTANTE

Recuerden que if tiene que funcionar si o si identado.

De momento solo hemos visto la condición que dice >= (mayor o igual), pero if tiene varias condicionantes, las cuales son:



IMPORTANTE

Recuerden que las comparaciones deben ser del mismo tipo, si estamos viendo mayor que o menor que, significa que es un número entonces la variable a comparar debe ser INT o FLOAT.

Otro ejemplo:

```
Edad = input("Ingrese su edad")
Edad = int(Edad)
if (Edad == 15):
```

```
        print(“ Felices Chambeland ”)
else:
        print(“Feliz cumpleaños”)
```

En este ejemplo se ve el comando de igualdad.

Pero pensemos que queremos colocar una nueva condición si no se cumple la primera, siguiendo el ejemplo, si la persona no tiene 15 ver si es menor o mayor de edad, para poder hacer esto existe comando ELIF (que es una mezcla de ELSE y IF), ejemplo:

```
Edad = input(“Ingrese su edad”)
Edad = int(Edad) // Recueden que al momento de comparar estamos comparando Numeros
if (Edad == 15):
        print(“ Felices Chambeland ”)
elif(Edad >= 18):
        print(“Feliz cumpleaños, siendo mayor de Edad”)
else:
        print(“Feliz cumpleaños siendo menor de Edad”)
```

5.2 Condiciones de IF

Digamos que quieres hacer un programa que tenga más de una validación para una ejecución. Ejemplo, quiero conducir un auto, para esta acción necesito ser mayor de edad y también tener licencia de conducir.

Para estos casos Python tiene el comando and (AND, en español “y”) y el comando or (OR, en español “o”).

5.2.1 Comando AND

El comando AND para poder entrar al if se tienen que cumplir todas las condiciones. Ejemplo: que tenga licencia Y sea mayor de edad. Con que no se cumpla solo 1 de estas condiciones el programa no entrará al ciclo del IF.

Ejemplo:

```
Edad = True
Licencia = False
if ( Edad >= True and Licencia == True):
        Print(“puedo manejar el automovil”)
else:
        print(“no puedo manejar el automovil”)
```

5.2.2 Comando OR

Sirve con que solo se cumpla 1 de las condiciones para entrar al ciclo del if. Ejemplo: Una persona puede ir a su trabajo en auto o en bicicleta, entonces si tengo un auto entro al ciclo if, o si tengo una bicicleta también entro al ciclo if, si tengo las 2 cosas igual entro al if, la única manera de no entrar al ciclo es que si no tengo un auto ni bicicleta.

Ejemplo:

```
Auto = False
Bicicleta = True
if( Auto == True or Bicicleta == True):
        print(“Puedo ir al trabajo por mi cuenta.”)
else:
        print(“Tengo que tomar locomoción colectiva.”)
```

Datos:

Al hacer un if, siempre se está buscando una validación True o False.

Si escriben en la consola de python, 10 >= 3, se darán cuenta que retornara un True, o si lo escriben al revés 3 >= 10 retornara un False.

Entonces por lo mencionado, si yo trabajo con una variable boleana, no se necesita poner el == True o == False, dado que poner solamente la variable funcionará, dado que vera solo si es True o False. Realizando el mismo ejemplo anterior con la sintaxis reducida.

```
Auto = False
Bicicleta = True
if( Auto || Bicicleta ):
        print(“Puedo ir al trabajo por mi cuenta.”)
else:
        print(“Tengo que tomar locomoción colectiva.”)
//Si ejecutan este código verán que es exactamente el mismo resultado que el anterior.
```

5.2 For

Piensen que tengo les pido que impriman hagan una impresión de 200 números correlativos, es decir de 1 hasta 200. Usted piensa que tendrá que escribir 200 veces Print(), bueno en cualquier tipo de programación se puede realizar ciclos, uno de esos es el For (que traducido a pseudo código seria Para.)

La estructura de For en Python3 es:

```
for variable_generada in (rango, cadena o lista):
        accion.
```

5.3 Range

Revisemos algunos ejemplos para comprenderlo:

Sigamos el ejemplo de imprimir 200 números seguidos, esto sería:

```
 for i in range(200):
        print(i)
// este código genera una variable "i", que solo puede ser utilizada dentro del for, esta variable va ir iterando, secuencialmente desde 0 a 199, debido que le dijimos que imprima 200 números. Pero digamos que ahora queremos imprimir valores desde 1 hasta 200. Seria de la siguiente forma.
for i in range(1,201):
        print(i)
// ahora se mostrarán números desde 1 hasta 200, debido que la instrucción es, que sea desde 1 hasta un numero menos que 201.
Por defecto range itera 1 en 1, pero digamos que ahora queremos que haga más saltos.

for i in range(0,201,2):
        print(i)
// la estructura de range seria (inicio, menor que, de a cuanto)
```

5.4 Cadena

Ahora queremos recorrer una cadena de texto. Ejemplo:

```
Palabra = "Hola mundo IPP"
for i in Palabra:
        print("la palabra en saltos seria separada en : " + i)
```

Lo que el computador mostraría seria:

```
la palabra en saltos seria separada en :H
la palabra en saltos seria separada en :o
la palabra en saltos seria separada en :l
la palabra en saltos seria separada en :a
```

```
la palabra en saltos seria separada en :
la palabra en saltos seria separada en :m
la palabra en saltos seria separada en :u
la palabra en saltos seria separada en :n
la palabra en saltos seria separada en :d
la palabra en saltos seria separada en :o
la palabra en saltos seria separada en :
la palabra en saltos seria separada en :I
la palabra en saltos seria separada en :P
la palabra en saltos seria separada en :P
```

Se le informa al programa que para cada carácter tomará el valor de i de la cadena de string, y esta comienza a iterar, como cada valor de i es Char, entonces se puede imprimir fácilmente.

5.5 Lista

Ahora queremos recorrer cada carácter de una lista.

Lista = [“pepe”,3232,”a”, True, 432,433,3,”IPP”]

Tenemos esta siguiente lista que es mixta con caracteres, números y operaciones booleanas. La queremos recorrer, se recorre de la siguiente forma:

```
for i in Nombre_Lista:
        accion()
```

Ejemplo:

```
for i in Lista:
        print(i)
```

El resultado entregado por el computador es:

```
pepe
3232
a
True
432
433
3
IPP
```

5.6 While

Ya sabemos recorrer un programa según la secuencia de datos ya determinada, pero digamos que queremos hacer un ciclo sin saber la cantidad de veces que se va a ejecutar, para esto existe while (que significa mientras), entonces le informamos al computador, que mientras ocurra una condición se ejecute.

Sintaxis:

```
while (condicion):
        accion()
```

Ejemplo:

```
Variable = 230
while(variable > 4):
        print (Variable)
        Variable = Variable / 3
```

Lo que el computador mostrara será:

```
130
43.333333333333336
14.444444444444445
4.814814814814815
```

¿Por qué no siguió? Porque la condición en el ejemplo era mientras la variable sea mayor a 4 entra al ciclo, entonces en la última iteración que entró (Variable), quedo con el valor de 1.6049 aprox. Y después comparo si Variable era mayor (>) que 4 para seguir cosa que no era cierta y por ese motivo se salió del ciclo.

Por lo visto anteriormente en la sección de IF, se darán cuenta que las comprobaciones entregan solo True o False. Entonces siguiendo esta lógica podríamos hacer un ciclo While (True), cuyo ciclo será infinito, pero un ciclo infinito sin ninguna condición de corte el computador y el compilador de Python lo botará o en algunos casos dejará nuestros computadores *pegados*. Para poder arreglar este motivo Python tiene la variable Break.

Break es una condición que se le a cualquier ciclo, para que este saga de este.

Ejemplo:

```
while(True):
   Variable = input("ingrese una variable \n")
```

```
Variable = int(Variable)
if (Variable == 0):
    break
print(Variable)
```

El computador mostrará:

```
ingrese una variable
32
32
ingrese una variable
43
43
ingrese una variable
23
23
ingrese una variable
231
231
ingrese una variable
0
```

En este código, se tuvo que costear la información a int, porque cuando realizábamos al comparación de Variable == 0, no entraba al ciclo ya que comparaba el número 0 (cero), con el char de “0”, que para el computador es muy diferente, por este motivo se tuvo que realizar el casteo de los datos.

Dato, en el input al final se le agrego \n, que es un salto de línea, para que al momento de ingresar el dato no esté al lado del texto si no que este abajo de este.

5.7 Continue

Continue se ocupa si durante un ciclo, una etapa de un ciclo no se quiere hacer nada, al momento de estar en una iteración el programa se encuentra con un continue, lo que hace omite todo lo que hace el ciclo durante ese turno, y sigue con el siguiente.

En el siguiente ejemplo se ve más claro:

```
for i in "Python3":
    if i == "h":
        continue
    print ("Letra actual :" + i)
```

El computador mostrará:

```
Letra actual :P
Letra actual :y
Letra actual :t
Letra actual :o
Letra actual :n
Letra actual :3
```

Si se percatan cuando ocurre que se encuentra en la letra h, el computador se salta completamente el print, es como que se salta todo lo que viene y sigue adelante con el ciclo.

Ejemplo de Continue en un While:

```
Variable = 5
while Variable > 0:
   Variable -= 1
   if Variable == 3:
      continue
   print ("Valor actual de la variable : " + str(Variable))
```

El computador mostrará:

```
Valor actual de la variable : 4
Valor actual de la variable : 2
Valor actual de la variable : 1
Valor actual de la variable : 0
```

5.8 Pass

Pass, es una variable pasiva que se ocupa cuando no quieres que se haga nada dentro de un ciclo.

Viendo los mismos ejemplos anteriores:

```
for i in "Python3":
   if i == "h":
      Pass
   print ("Letra actual :" + i)
```

El resultado que entrega el computador será:

```
Letra actual :P
Letra actual :y
Letra actual :t
Letra actual :h
Letra actual :o
Letra actual :n
Letra actual :3
```

Se dan cuenta que no hizo nada, solo se ocupó para que no quede sin nada que hacer el if.

Ejemplo de Continue vs Pass:

```
print("for con continue")
for x in "Hola":
   print (x)
   continue
   print (str(x) + " denuevo :D ")

print("for con pass")

for x in "Hola":
   print (x)
   pass
   print (str(x) + " denuevo :D “)
```

El computador nos enmostrará:

```
for con continue
H
o
l
a
for con pass
H
H denuevo :D
o
o denuevo :D
l
l denuevo :D
```

a
a denuevo :D

6. Cierre

La importancia de tener un manejo más concreto de los datos puede verse aplicado en: recorrer una palabra, saber si esta tiene un número o letra en ella, crear una cadena lista de datos, etc.

Este tipo de datos tomará mucha fuerza al comenzar a trabajar cada vez más con el lenguaje de programación, en este caso es Python; pero el aprendizaje obtenido en este módulo les será de mucha utilidad tanto en este lenguaje de programación como en cualquier otro, evidenciando que el cambio será solo la sintaxis.

Al igual que en el módulo anterior, al momento de unificar una actividad en ellas, se obtenía siempre: “si ocurre, esto hago esto”, “realizo esta actividad mientras ocurra esto”, “repito esto tanta veces”. Ahora que tenemos el manejo de condiciones y también de ciclos, todo el análisis que logramos en el módulo anterior de descomposición, ahora la podremos llevar a la programación.

APORTE A TU FORMACIÓN

El estudiante tendra una mayor comprencion en el analisis de cualquier tipo de circunstancia que se encuentre tanto este relacionada con el ambito de la programacion o no, apliando su vision al momento de tomar cualquier decisión en su vida laboral