# Relatoría 1: Informática III

Juan Esteban Salgado

## Objetivos

- Identificar y aprender las herramientas que brinda el lenguaje de programación *python.*
- Aplicar este lenguaje de programación para solucionar problemas de interés aplicados a las matemáticas o física.
- Presentar un proyecto final en el cual se evalúen todas las competencias enseñadas durante el curso.

## Marco Teórico

### *Clase 17-02-2023*

- ***Configuración del Entorno de Trabajo***

Para comenzar con un lenguaje de programación se requieren de ciertas adecuaciones al entorno de trabajo para poder aprenderlo mejor. Para esto, como entorno gráfico se maneja la aplicación *"Visual Studio Code"*, en donde se redacta en cada clase ciertas directrices del trabajo a realizar, además de que esta aplicación es capaz de crear aplicaciones tipo *".py"*, las cuales son de vital importancia para empezar a programar en *python.*

Esto como entorno gráfico, pero también surge la necesidad de organizar la información en repositorios o carpetas de acceso online, en las cuales se pueda estar actualizando y posteando los códigos o anotaciones que se tomen en clase. Para realizar esta labor se maneja la página web de *"Github"*, aquí es sencillo enlazarlo con la aplicación *"Visual Studio Code"* brindando así la capacidad de tener nuestro trabajo en la nube. Cabe resaltar que para utilizar este entorno virtual, es necesario crear una cuenta y descargar una herramienta llamada *"Git"* para poder subir nuestro trabajo cómodamente.

- **Visual Studio Code:** "Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código." [1]

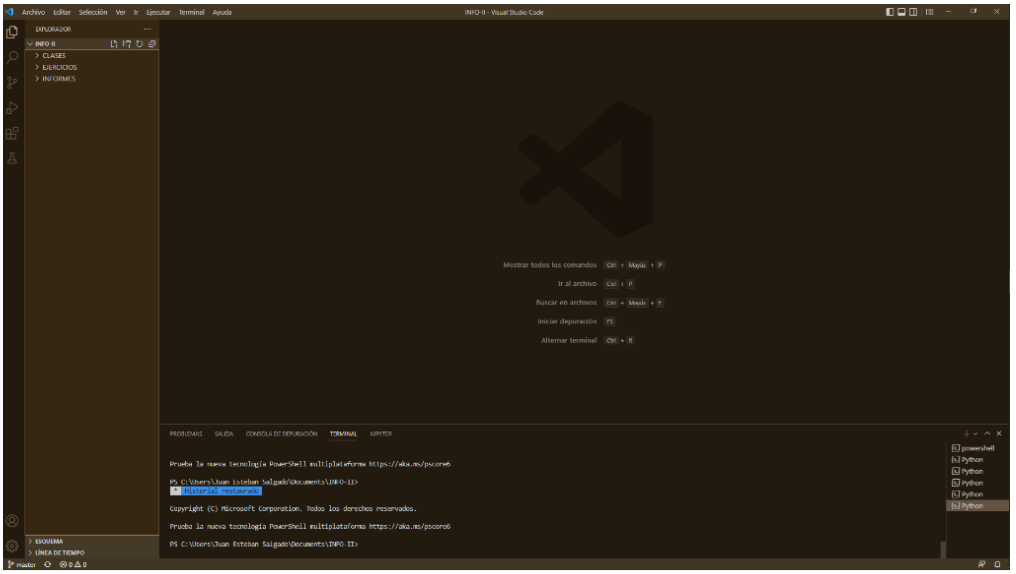


*Figura 1.* Entorno de desarrollo VSC.

- **GitHub:** “GitHub es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de ordenador.” [2]

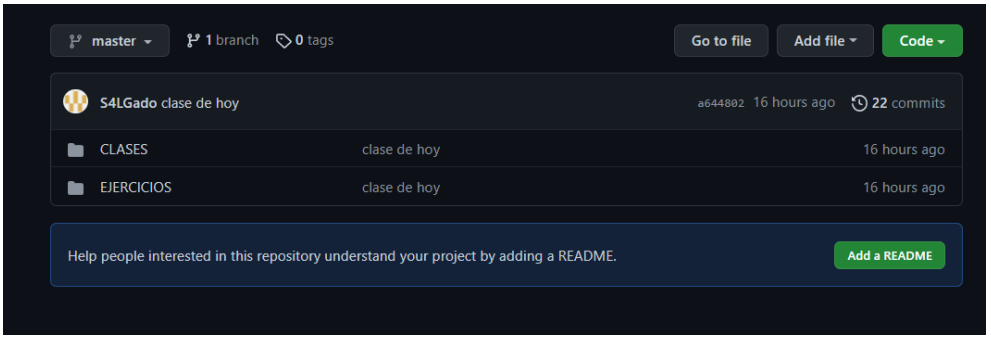


*Figura 2.* GitHub y los repositorios creados.

## *Clase 24-02-2023*

- ***Creación de Repositorios***

Para subir de forma adecuada nuestros proyectos a la nube, es de vital importancia crear un *repositorio*. Un repositorio se define como una serie de carpetas contenedoras de código o apuntes, las cuales se pueden modificar abiertamente o de forma privada y en las cuales se consignará el trabajo realizado durante el periodo estudiantil. Para crearlo simplemente se debe crear una *carpeta* windows y a través de VSC  (Visual Studio Code) se selecciona como repositorio.

Luego de seleccionada se deben hacer unas configuraciones en el terminal de VSC, ya que para que se actualice nuestro trabajo en la nube de GitHub se le debe dar los permisos necesarios al entorno de trabajo. Un *terminal* se define como el lugar en el cual se ejecutan los programas que se van realizando. Se debe escribir lo siguiente:

- **< git config --global user.name "Usuario" >**
- **< git config --global user.email "Correo" >**

Ya con esto es suficiente para comenzar a trabajar en nuestros proyectos, el único detalle que puede ser importante mencionar es a la hora de subir los archivos a GitHub. Para esto nuestros archivos deben pasar por tres fases

- **Changes:** Esta fase consiste en un cambio realizado al código que simplemente ha sido guardado.
- **Staged:** Aquí se confirman los cambios realizados para no subir una versión errónea.
- **Commit:** Simplemente se da la orden de subir los archivos a la nube, para esto es de vital importancia escribir un mensaje corto describiendo el trabajo realizado o en su defecto en qué consiste la actualización enviada.

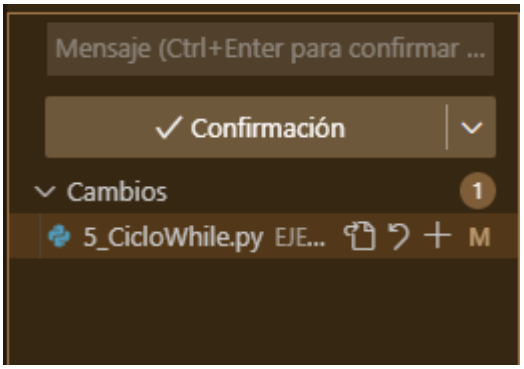


*Figura 3.* Sección de changes, staged y commit en VSC.

- ***Fundamentos de Python***

Para comenzar con cualquier lenguaje de programación se deben de revisar las características principales del mismo, una que resalta bastante es que *python* es un lenguaje de alto nivel, lo que significa que contiene bastante funciones integradas que pueden ayudar a resolver los problemas de forma más eficaz o rápida. Algunos ejemplos de estas funciones pueden ser: Suma, Resta, Vectores, Condicionales Etc…

Además es un lenguaje *interpretado*, lo que significa que a la hora de ejecutar un programa el lee línea por línea, avisando entonces si hay un error o detenerse en una línea específica.

Posee un *tipado dinámico,* lo cual le da más rigurosidad al lenguaje pues es necesario definir cada tipo de variable o de dato. A continuación se definirán los *tipos de datos:*

- **Tipos de datos**
- **Booleano:** Solo toma valores binarios como verdadero o falso ( true, false)
- **Entero:** Números enteros (1, 2, 3).
- **Flotante:** Números racionales (1.2, 3.6).
- **Strings:** Cadenas de caracteres (“hola”, “avión”).
- **Listas:** Serie de elementos cambiables ((1, “hola”)).
- **Tuplas:** Serie de elementos inmutables ([1, “hola”]).
- **Diccionarios:** Permite realizar un mapeo entre dos elementos ((clave:valor), (clave 2:valor)).
- **Conjuntos:** Solo admite valores en una serie de elementos {1,2,3}.

Es claro que para python todo elemento es diferente según como se describe, para esto también existen funciones capaces de cambiar el tipo de dato que se está manejando para que sea más sencillo de tratar.

- **Funciones de conversión de datos**

- **Int:** Convierte un dato en entero.

- **Float:** Convierte un dato en flotante.

- **Str:** Convierte un dato en string (cadena de caracteres).

```
# un entero a un flotante

a = float (a)
print (a, " es del tipo : ", type(a) )

# un flotante a un entero

e = int (e)
print (e, " es del tipo : ", type(e) )

# un string a un entero y flotante

h = int (h)
print (h, " es del tipo : ", type(h) )

h = float (h)
print (h, " es del tipo : ", type(h) )

# un número a un string

a = str (a)
print (a, " es del tipo : ", type(a) )
```

*Ejemplo 1.* Manejo de las funciones de conversión de datos.

Cabe resaltar que si un dato no cumple con las condiciones no se puede pasar de un tipo a otro, por ejemplo una letra “a” no se puede volver entero o flotante, pues python no sabría describir esta variable numéricamente.

## *Clase 01-03-2023*

- ***Operadores***

Es claro que para resolver un problema se han de necesitar ciertas herramientas matemáticas, algunas básicas y otras un poco más específicas que contiene el lenguaje de python. Los operadores se dividen en diferentes tipos, los cuales se describirán a continuación

- **Tipos de Operadores**
- **Asignación:** = (se usa para declarar una variable).
- **Aritméticos:** + (suma), - (resta), * (multiplicación), / (división), // (división entera), % (residuo de una división), ** (potenciación).

```
# Operadores ariméticos
""" 3 + 9
3.0 + 9
9 ** 0.5
2 ** 32
19 // 2
19 % 3 ***
"hola" + "mundo"
"hola" * 3
["A"] + [1,2,3]
[] + []
(1,2,3) + (1,) """
```

*Ejemplo 2.* Operadores aritméticos.

- **Lógicos:** and (y), or (o), not (no)

```
# Operadores Lógicos
""" True and True
False and True
False and False
not True
not False
True or True
False or True
1 and 1
0 and 1
1 and 3
1 and "hola"
0 and 3
0 and "hola"
"hola" or "verdadero"
1 or 3"" or "hola"
"" or "false" """
```

*Ejemplo 3.* Operadores lógicos.

- **Comparación:** > (mayor que), >= (mayor o igual que), < (menor que), <=(menor o igual que), == (igual), != (diferente de).

```
# Operadores de comparación
""" 1 > 2
1 < 3
1 == 1
2 != 1
3 >= 3
5 <= 2
4 > True
True > False
[] > [1,2,3]
"a" > "b" """
```

*Ejemplo 4.* Operadores de comparación.

- **Pertenencia:** in (adentro), not in (afuera).

```
# Operadores de pertenencia

""" "a" in "abcdefg"
"A" in "ABCDEFG"
1 in [1, 2, 3]
1 in ["1", "2", "3"]
"hola" in "holamundo"
"Hola" not in "holamundo" """
```

*Ejemplo 5.* Operadores de pertenencia.

- **Conjuntos:** | (unión), & (intersección), - (diferencia).

Tener claro estas definiciones es vital, ya que así se sabe cuál operador es el correcto en cada caso. Cabe resaltar que al utilizar operadores de comparación la respuesta va a ser de tipo booleana, esto es importante ya que a la hora de utilizar condicionales, se puede saber si se cumple o no con la condición dada, para realizar la acción.

## *Clase 01-03-2022*

- ***Funciones Integradas***

Como fue mencionado anteriormente, python se caracteriza por ser un lenguaje de alto nivel, por lo que ahora se describirán las funciones integradas en este lenguaje que facilitan ampliamente el uso del mismo. Además de que brinda un abanico de posibilidades para realizar operaciones de forma rápida y eficaz. A continuación se mencionan algunas de estas funciones y sus aplicaciones:

- **Entrada y Salida:** Sirven para que el usuario pueda ingresar o observar un mensaje en el terminal, Input (entrada de un dato), Print (escribe un mensaje deseado en el terminal), Format(cambia el formato de escritura de un mensaje o variable).

```
"""Edad = int(input("Digite su edad: "))
if Edad >= 18:
| print ("Es mayor de edad")
if Edad < 18:
| print ("Es menor de edad")"""
```

*Ejemplo 6.* Ejercicio con manejo de input y print.

```
"""numero1 = 0.52941

print(format (numero1, ".0f"))
print(format (numero1, ".2f"))
print(format (numero1, ".5f"))
print(format (numero1, ".1e"))
print(format (numero1, ".2e"))
```

*Ejemplo 7.* Ejercicio con manejo de format.

- **Ayuda:** Se utilizan para obtener información sobre algún valor o variable, esto con el fin de darle un mejor manejo. Help (da información acerca de un módulo, clase, función o variable en específico), Dir (entrega todas las propiedades del objeto en concreto), Type (indica el tipo de dato que es una variable).
- **Matemáticas:** En este apartado hay bastante funciones, pero las más destacables podrían ser, Abs (valor absoluto), Round (redondea a una cantidad de decimales), Pow (eleva una variable a otra).
- **Conversiones:** Este tipo de funciones ya fueron descritas cuando se mencionaron los tipos de datos, pero a parte de Int, Float y Str. Hay otras más específicas como lo puede ser Complex (convierte un número real en imaginario), Bool (entrega un booleano a partir de una variable cualquiera), List (define un conjunto de datos como

una lista para ser trabajada matemáticamente). Hay una subdivisión muy importante a la hora de las conversiones y esta es entre sistemas numéricos:

- **Bin:** Convierte un número real en binario.
- **Oct:** Convierte un número real en octal.
- **Hex:** Convierte un número real en hexadecimal.

```
# Convertir a binario, octal y hexadecimal
"""
decimal = 9
conversion_binario = bin(decimal)
conversion_octal = oct(decimal)
conversion_hex = hex(decimal)

print(conversion_binario)
print(conversion_octal)
print(conversion_hex ) """

# ¿ Cómo hacer lo contrario ?
"""
bin, oct, hex = "1100110", "146", "66"

print (" bin a decimal: ", int(bin,2))
print (" octal a decimal: ", int (oct,8))
print ("hexadecimal a decimal: ", int(hex,16)) """
```

*Ejemplo 8.* Manejo de la conversión entre sistemas numéricos.

- **Secuencias:** Estas se utilizan para secuencias numéricas o de variables cualesquiera, Range (genera números enteros hasta que se le de una parada), Enumerate (se utiliza cuando se necesita el valor de un iterable), Zip (entre un objeto comprimido).
- **Operaciones en Secuencias:** a veces es necesario alterar el orden de las secuencias o en su defecto saber algunas propiedades de las mismas. Len (entrega el número de elementos en una secuencia), Sum (entrega la suma de los números en la secuencia), Max (entrega el valor máximo de la secuencia), Min (entrega el valor mínimo de la secuencia).

```
secuencia7 = range(1,10000,3)
lista = [1,2,3,4,5,8,8,9]

print("Tamaño de secuencia", len(secuencia7))
print("Tamaño de lista", len(lista))

print("Mínimo de secuencia", min(secuencia7))
print("Mínimo de lista", min(lista))

print("Máximo de secuencia", max(secuencia7))
print("Máximo de listaa", max(lista))

print("Revertir secuencia", list(reversed(secuencia7)))
print("Revertir lista", list(reversed(lista)))
```

*Ejemplo 9.* Uso de secuencias y operaciones en secuencias.

Estas son algunas de las operaciones básicas, cabe mencionar que no es necesario llamar a ninguna *librería* para el uso de estas. Más adelante se definirá lo que es una librería y cómo puede traer funciones nuevas que el lenguaje de python no contiene.

## *Clase 03-03 -2023*

- ***Condicional If***

Este es el condicional más básico de cualquier lenguaje de programación como C + + o MatLab y en el caso de Python maneja la misma funcionalidad. Este condicional consiste en verificar si una condición dada es verdadera o falsa y luego de eso realizar una serie de tareas por haber cumplido la condición inicial. En caso de que la condición sea falsa simplemente no se ejecuta ninguna acción y se sigue con el resto de líneas del código.

```
if <condicion>:
    <sentencias>
    <sentencias>
    <sentencias>
    <sentencias>

elif <condicion2>:
    <sentencias>
```

*Figura 4.* Esquema de uso del condicional if.

```
# Ejercicio 1
"""
Pida a un usuario su nombre y su edad. Determine si es mayor de edad,
y muestre un mensaje en pantalla diciendo:
<NOMBRE>, usted es mayor/menor de edad:
"""
"""
nombre = input("INgrese su nombre: ")
edad = int(input("Ingrese la edad: "))

if edad >= 18:
    print(nombre, "{}, usted es mayor de edad", format(nombre))
elif 0 < edad < 18:
    print(nombre, "{}, usted es menor de edad", format(nombre))
"""
# Ejercicio 2
"""
Realice un programa que calcule el mayor de tres numeros
"""
"""
a = float(input("Digite el numero 1: "))
b = float(input("Digite el numero 2: "))
c = float(input("Digite el numero 3: "))

if a >= b and a >= c:
    mayor = a
elif b >= a and b >= c:
    mayor = b
elif c >= a and c >= b:
    mayor = c

print ("El mayor numero entre los ingresados es: ",mayor)

"""
```

*Ejemplo 10.* Dos ejercicios diferentes en donde se puede aplicar el condicional IF.

- ***Ciclo While***

Es bastante similar al método *if*, sin embargo es cíclico hasta que se deje de cumplir la condición dada. Claramente si la condición es tomada como verdadera (True) va a continuar realizando la acción, en caso contrario (False) dejará de realizar la acción inmediatamente.

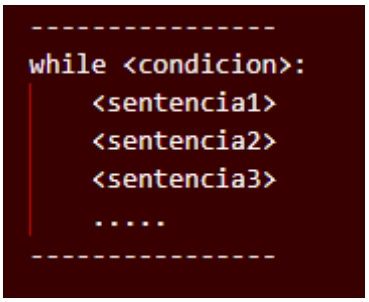


*Figura 1*. Esquema del uso del condicional *While*.

Un ejemplo muy básico del uso del ciclo *While*, puede ser el siguiente descrito en el Ejemplo 1, donde simplemente se tiene un contador que hasta llegar a cierto valor ( en este caso 100 ) va a detener el ciclo. Para detener el ciclo se recalca la importancia de hacer que la condición inicial sea falsa de lo contrario puede ser un ciclo infinito.

```
condicion = True
contador = 0
while condicion:
    print("ciclo ejecutado {}".format(contador))
    contador = contador + 1
    if contador == 100:
        break
```

*Ejemplo 1*. Ejemplo básico del uso del condicional While.

Este ciclo es de vital importancia ya que define lo que son *funciones de recursión,* lo cual matemáticamente hablando supone un gran avance en términos de programación, ya que uno de los usos que más se le dan a los lenguajes de programación es para métodos matemáticos

aproximados, y estos van de la mando con funciones recursivas en las que se utiliza un valor anterior para calcular el siguiente que a su vez tiene un porcentaje mucho menor.

## *Clase 15-03-2023*

- ***Ciclo For***

Uno de los ciclos más importantes para recorrer listas o elementos en un diccionario. Funciona muy diferente a como lo hace el ciclo *For* en otros lenguajes de programación. En el caso de Matlab y C + + el ciclo consiste en definir un iterable que va a realizar una acción cierta cantidad de veces. Además de que este iterable puede hacer parte de alguna fórmula o recorrer un vector mientras esté dentro del ciclo.

En el caso de Python esto va más allá, ya que el iterable se puede definir desde un número hasta los mismo elementos de una lista. Esta flexibilidad le brinda al programador mucha libertad para recorrer listas o hasta incluso un string, todo con la definición del iterable y del elemento que se quiera recorrer. A diferencia del ciclo anterior (while), se puede decir que para este ya debemos tener un final definido, porque el ciclo al recorrer todo el elemento se detendrá.

```
for <variable> in <iterable>:
    <secuencia1>
    <secuencia2>
    <secuencia3>
```

*Figura 2.* Esquema del uso del ciclo *For*.

Una de las aplicaciones innovadoras de recorrer un iterable con el ciclo For. Consiste en poder recorrer una cadena (string) de forma muy sencilla, pudiendo analizar elemento a

elemento, donde la variable que nosotros definamos irá cambiando respecto a cada letra que contenga esta cadena.

```
cadena = "HolaMundoCruel"
lista = [1,2,30,100,50,-20]
print("-Recorrido de una cadena-")
for caracter in cadena :
    print(caracter,end="--") #Escribe cada caracter de la cadena por separado
print ("\n-Recorrido de una lista")
for i in lista:
    print(i, end="--")
```

*Ejemplo 2.* Manejo del ciclo For para un string y para una lista

En el caso del Ejemplo 2 se puede apreciar cómo se recorre un string y también una lista, para este ciclo se determinó una escritura en la cual se separaría cada variable por un "--", hay muchas aplicaciones en las cuales se pueden establecer por ejemplo contadores que al recibir cierta cantidad de letras iguales o cantidad de elementos iguales puedan operar. Lo importante que se debe resaltar sobre este ciclo, es que el nombre de la variable realmente no es de relevancia (aunque generalmente se utilice un nombre relacionado al iterable que se va a recorrer), sino lo que es de gran importancia es el iterable que se va a recorrer, ya que nuestra variable va a cambiar según el tipo de datos que se tenga dentro del iterable. Dando un ejemplo muy claro: Si recorro una lista que contiene listas, mi variable van a ser listas.

.

También si se le quiere dar un uso al For un poco más convencional se puede recurrir a los datos tipo "Range" en los cuales se puede describir una serie de números que se deben de recorrer para que este sea de uso más mecánico como en otros lenguajes de programación.

```
for i in range (0,21,2):
    print (i, end=" ")
```

*Ejemplo 3.* Uso del ciclo for para escribir una serie de números.

El Ejemplo 3 es un claro uso de un *Range* con el cual el iterable va  a recorrer estos números en este caso desde el 0 hasta el 20 con pasos de 2 en 2. Es muy útil esto para recorrer ciertos términos en una lista o en un diccionario y poder extraer los pares o impares.

## *Clase 17-03-2023*

- ***Métodos***

Python al ser un lenguaje de programación tan avanzado, es capaz de realizar ciertas modificaciones a los datos que se tengan en el mismo. Estas modificaciones son llamadas *métodos*. Para acceder a estos métodos basta con escribir un "." después de nuestra variable y según el tipo  de dato que tenga se le pueden realizar diferentes acciones, así que se mostrarán algunos ejemplos de estas:

- ***Strings***

  **Formateo:**

- **Capitalize:** Crea una copia del string y coloca la primera letra en mayúscula.
- **Upper:** Coloca todas las letras en mayúscula.
- **Lower:** Coloca todas las letras en minúscula.
- **Center:** Alinea el string en el centro.
- **Title:** Entrega un string donde la primera letra de cada palabra está en mayúscula.
- **Strip:** Remueve cualquier espacio entre las palabras.

```
print(cadena.upper())
print(cadena.lower())
print(cadena.count("u"))
print(cadena.isalpha())
print(cadena.isalnum())
print(cadena.isnumeric())
print(cadena.replace("u","i"))
```

*Ejemplo 4.* Uso de métodos de formateo en strings.

### Operaciones:

- **Count:** Cuenta el substring que se pida.
- **Find:** Entrega el índice en que se encuentra el substring que se pida.
- **Replace:** Entrega otro string con todos los substring reemplazados por otro.

### Verificación:

- **Isalnum:** Determina si es alfanumérico.
- **Isalpha:** Determina si es alfabético.
- **Isdigit:** Determina si solo contiene números.
- **Isdecimal:** Determina si todos los términos son números decimales.

### Tipos de datos:

- **Booleano:** Solo toma valores binarios como verdadero o falso ( true, false)

### Indexado:

- **[Índice]:** Generalmente cuando se recorre un string se usa el corchete para definir qué índice se quiere revisar.

### Slicing:

- **[Indice 1:Indice 2:salto]:** Similar al indexado ahora se recorre el string según los parámetros dados, muy parecido a una variable tipo Range.

```
# Indexación de Cadenas

cadena2 = "Mundo Unal"

print (cadena2[0: 10: 2])

# Cadena al revés

print(cadena2[-1::-1])

# Elementos ubicados en las posiciones 3,5 y 7

print(cadena2[2:8:2])

# Elementos hasta la mitad de la cadena

print(cadena2[0:(len(cadena2)//2):1])

# Elementos de la mitad en adelante
print(cadena2[(len(cadena2)//2):len(cadena2):1])

cadena3= "Anita no lava la tina de lunes a viernes"
```

*Ejemplo 5.* Uso de indexaciones y slicings para strings.

- ***Listas***

  **Operaciones:**

- **Append:** Se usa para ingresar un nuevo término a la lista.
- **Insert:** Ingresa un nuevo término a la lista pero en un índice específico.
- **Pop:** Remueve de la lista el término en el índice deseado.
- **Remove:** Remueve el término deseado de la lista sin importar el índice.
- **Count:** Cuentas cuantas veces se repite un término.

```
lista = [1,'a',2,3,'b']

lista.append('1000') # Agrega un valor la lista
lista.append(10)
lista.insert(5,10) # Inserta un valor en un indice deseado (indice,valor)
lista.pop(0) # Indica que indice eliminar
lista.insert(4,'Salgado')
```

*Ejemplo 6*. Manejo de Métodos operacionales para listas.

### Ordenado:

- **Sort:** Organiza la lista de menor a mayor, en caso de ser string lo hace por orden alfabético.
- **Reverse:** Revierte el orden de una lista.

### Almacenamiento:

- **Clear:** Borra todos los elementos de una lista.
- **Copy:** Genera una copia de la lista, esto es de vital importancia en python ya que al definir dos variables iguales, estas van a cambiar de la misma manera.

### Indexado:

- **[Índice]:** Generalmente cuando se recorre una lista se usa el corchete para definir qué índice se quiere revisar.

### Slicing:

- **[Indice 1:Indice 2:salto]:** Similar al indexado ahora se recorre la lista según los parámetros dados, muy parecido a una variable tipo Range.

```python
# Metodo indexado y slicing

listaNueva = [1,2,3,4,5,'hola','cruel','mundo',100]

# Extraer el primer elemento de 2 maneras

print(listaNueva[0],listaNueva[-9])

# Extraer el ultimo elemento de 2 maneras

print(listaNueva[8],listaNueva[-1])

# Extraer el elmento de la mitad de 2 maneras

print(listaNueva[4],listaNueva[-5])

# Slicing

# Extraer cada elemento de dos en dos
print(listaNueva[0:9:2]) # Otra forma de llegar al final es [0::2]
# Extraer hasta la mitad de la cadena
print(listaNueva[0:5:1])
# Extrear desde la mitad de la cadena en adelante
print(listaNueva[5::1])
# Extraer los elementos que son strings
print(listaNueva[5:8])
# Extraer los elementos que son enteros
print(listaNueva[0:5] + [listaNueva[-1]])
```

*Ejemplo 7.* Métodos indexados y slicings para listas.

- ***Tuplas***

  Las tuplas manejan solamente operaciones como index y count. Ya que por su naturaleza no se pueden alterar fácilmente como con una lista. También pueden ser indexados y realizarles slicing.

- ***Diccionarios***

  **Extracción:**

- **Items:** Entrega tanto clave como valor del diccionario.
- **Keys:** Entrega las claves de un diccionario.
- **Values:** Entrega los valores de un diccionario.
- **Get:** Entrega el valor de la clave que se esté buscando.

  **Eliminar:**

- **Pop:** Remueve el valor para cierta clave determinada.

  **Almacenamiento:**

- **Clear:** Borra todo el diccionario.
- **Copy:** Crea una copia del diccionario.

  **Indexado:**

- **[Índice]:** Generalmente cuando se recorre un diccionario se usa el corchete para definir qué índice se quiere revisar.

## *Clase 22-03-2023*

- ***Funciones***

Es claro que a veces es necesario tener pequeñas definiciones que permitan simplificar en gran manera un código. Es por esto que se crean las funciones, para permitir al programador acceder a definiciones que hizo anteriormente y evitar reescribir un proceso. Sin importar en donde se ubiquen estas pequeñas secciones van a ser llamadas a conveniencia.

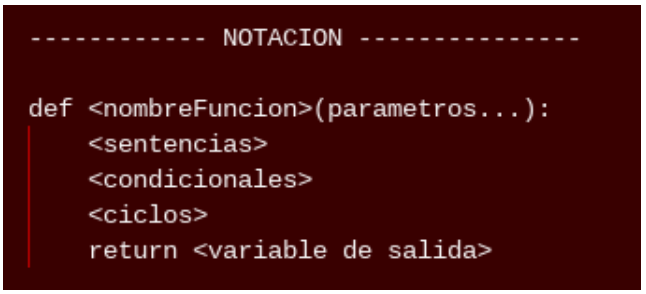


*Figura 3.* Esquema de una función.

Como se puede observar en la figura 3, es claro que cada que se haga una función se debe de retornar algo. Es muy importante que este elemento retornado sea una variable y no una impresión en pantalla como usualmente se ve. Además las sentencias de la función se van a repetir según el programador lo desee. A continuación unos ejemplos del manejo de funciones y sus diversas aplicaciones.

```
"""
1) Desarrollar una funcion que reciba dos numeros y
devuelva la suma de ambos
"""

def suma(a,b):
  resultadoSuma = a + b
  return resultadoSuma

"""
2) Desarrollar una funcion que reciba dos listas y devuelva una nueva lista,
que sume elemento a elemento
"""

def sumaListas(c,d):
  resultadoSumaListas = []
  for i in c:
    valor = i + d[c.index(i)]
    resultadoSumaListas.append(valor)
  return resultadoSumaListas

"""
3) Desarrollar una funcion que no reciba parametros
y que no tome valores, pero que sirva para imprimir un mensaje
de 3 lineas
"""

def mensaje3Lineas():
  Lineas = "\n \n \n"
  return Lineas

"""
```

*Ejemplo 11.* Tres ejemplos de funciones básicas.

Cabe resaltar que las aplicaciones mostradas en el ejemplo 11 son muy básicas, pero vuelven a la premisa inicial y es resumir en forma considerable la cantidad de código y procesamiento que se debe realizar. Una ventaja de Python es que definir y llamar sus funciones es muy fácil en comparación a otros lenguajes de programación.

# Bibliografías o Webgrafías


- [1] Lardinois, Frederic (29 de abril de 2015). «Microsoft Launches Visual Studio Code, A Free Cross-Platform Code Editor For OS X, Linux And Windows». TechCrunch.
- [2] «Microsoft Investor Relations - Acquisitions History». www.microsoft.com (en inglés).
- [3] «Curso de Python». https://codigofacilito.com/cursos/Python