Una Ambiciosa Introducción a Python
Parte II

print(“Bienvenidos a AIPython II”)

Las funciones en Python son mas flexibles que en matemáticas

- Permiten causar algún efecto, por ejemplo enviar texto a la terminal,etc.
- Evaluar un valor, calcular la potencia de un numero,etc.

Argumentos
- Van entre los paréntesis.
- Pueden ser una cantidad de objetos arbitrarios.

Invocación
- Es el nombre de la función los paréntesis y la lista de argumentos.

Por ejemplo la función print
- Su efecto: mostrar lo que se pasa por argumento.
- Sus argumentos: espera cualquiera, puede operar con cualquier tipo de datos (literales)
- Que valor retorna: ninguno (None), su efecto es suficiente.

Argumentos Posicionales print(“Bienvenidos”, “al curso”, “de Python”)
- Están definidos por la posición en la invocación y separados por como (,)

Argumentos de palabras claves
- El significado proviene no por su posición, sino de palabras claves utilizadas para identificarlos.
- Por ejemplo la función print tiene dos, end y sep.
- La regla de oro dice deben ir luego de los argumentos posicionales y constan de el nombre el signo igual y un valor.

Carácter de escape
La barra invertida, permite indicarle a la función print dentro del texto que pasamos como argumento, que reaccione de otra manera a cierta combinación de caracteres.

- n, significa que se debe agregar una nueva línea luego de nuestro texto, la n proviene de nueva línea (new line).’
- \\, indica que escape del carácter y lo tome de manera exacta, saldrá \ en pantalla.
- t, deja 8 espacios.

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Datos en Python - Literales “Se refiere a datos cuyos valores están determinados por el mismo literal”

- Enteros int , aquellos que no tienen parte fraccionaria.
- Punto flotante float, son capaces de contener una parte fraccionaria.
- Cadenas str, se emplean cuando se requiere procesar texto. Son aquellos encerrados entre comillas.
- Booleans True, False, hace uso de dos valores refiere a estados, nos ayudaran a responder preguntas que los programadores hacen en sus programas.

Operadores Básicos

- Aritméticos +,-,*,/,//,%,**

El orden de prioridad sería el siguiente para los operadores aritméticos, siendo el primero el de mayor prioridad:

1. Paréntesis ()
2. Exponente **
3. Negación -x
4. Multiplicación *, División /, Cociente //, Módulo %
5. Suma, Resta, +, -

Variables

Nos permiten almacenar objetos (literales por ahora), resultados, etc. Tienen un nombre y un valor que se almacena en el contenedor.

- Deben comenzar con una letra o con un guion bajo (_)
- Mayúsculas y minúsculas se tratan de manera diferente.
- No pueden llamarse como alguna de las tantas palabras reservadas del lenguaje Python.

Válidas

nombre, apellido, edad, país, primer_nombre, fecha_nacimiento, _if

Invalidas

primer-nombre, primer@nombre, primer$nombre, num-1, 2num

Operador asignación =

• Este operador tiene una sintaxis muy simple y una interpretación casi natural. Asigna el valor del argumento de la derecha al de la izquierda.

```
edad=34
```

Operadores Abreviados

```
variable = variable operador expresión
#su contracción
variable operador = expresión
var = var / 2
var /= 2
```

Comentarios #

Es un texto que es ignorado por el interprete de Python #soy un comentario

En python no existen comentarios de múltiples líneas. Si queremos comentar en variables líneas, cada una de estas debe comenzar con un #.

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Convenciones PEP

Cuando escribimos código en Python, no solo importa que funcione, sino también que sea legible, claro y mantenible. Para lograrlo, la comunidad de Python sigue una serie de recomendaciones llamadas PEP (Python Enhancement Proposals).

Una de las más importantes es la PEP 8, que establece convenciones sobre:

✅ Formato del código: Indentación con 4 espacios, líneas de hasta 79 caracteres.
✅ Nombres de variables y funciones: snake_case para funciones y variables, CamelCase para clases.
✅ Espaciado y organización: Evitar espacios innecesarios y usar líneas en blanco para separar secciones.
✅ Comentarios y documentación: Explicar el código de manera clara con # o docstrings (""" """).

Seguir estas convenciones hace que nuestro código sea más fácil de leer y comprender, tanto para nosotros como para otros programadores.

```
def Sumar(a,b): return a+b
def restar(a,b): return a-b
def multiplicar(a,b): return a*b

def main():
 x=int(input("Ingrese un numero:"))
 y=int(input("Ingrese otro numero:"))
 print("Suma:", Sumar(x,y))
 print("restar:", restar(x,y))
 print("producto:", multiplicar(x,y))

main()
```

Nombres de funciones inconsistentes
(Sumar con mayúscula, restar con minúscula).

Falta de espacios después de comas.

Bloques de código sin indentación adecuada ni líneas en blanco.

main() sin espaciado entre definiciones.

Variables con nombres poco descriptivos (x, y).

```
def sumar(a, b):
    """Devuelve la suma de dos números."""
    return a + b


def restar(a, b):
    """Devuelve la resta de dos números."""
    return a - b


def multiplicar(a, b):
    """Devuelve el producto de dos números."""
    return a * b


def main():
    """Función principal que solicita dos números y
muestra los resultados."""
    num1 = int(input("Ingrese un número: "))
    num2 = int(input("Ingrese otro número: "))

    print("Suma:", sumar(num1, num2))
    print("Resta:", restar(num1, num2))
    print("Multiplicación:", multiplicar(num1, num2))


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

Nombres de funciones en minúscula y con verbos descriptivos.

Docstrings para explicar cada función.

Espaciado correcto entre definiciones. Dos líneas
L
Variables con nombres descriptivos (num1, num2 en lugar de x, y).

Uso del bloque if __name__ == "__main__":
para ejecutar main(), buena práctica en scripts.

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Función input

- Nos permite obtener datos de la consola, a diferencia de print, input nos regresa un valor que utilizaremos
- Puede recibir un argumento, para mostrar un texto en la terminal.
- El resultado debe asignarse a una variable ó se perderá.
- El resultado es siempre una cadena.

Casting, Python nos ofrece funciones para castear un tipo de dato:

- int()
- float()
- str()

Todas estas funciones toman un argumento y trata de convertirlo, sino puede, dará un error.

Operadores +, * en cadenas

- Además de sumar y multiplicar, estos operadores son capaces de concatenar y replicar respectivamente.
- print("cur"+"so") # curso
- resultado = "curso"*3
- print(resultado) #cursocursocurso

Otros métodos sobre cadenas

- title(), devuelve una copia de la cadena donde las palabras comenzaran con mayúsculas y el resto en minúsculas.
- upper(), devuelve una copia de la cadena con todos los caracteres en mayúsculas.
- lower(), devuelve una copia de la cadena con todos los caracteres en minúsculas.
- replace(), devuelve una copia de la cadena cambiando todas las ocurrencias de la su cadena por la nueva.
- len(), devuelve la cantidad de elementos que tiene la cadena pasada como argumento.

…… y muchos mas!!!

Operadores, expresiones, comparación

Operador	Significado
>	Mayor
<	Menor
>=	Mayor o igual
<=	Menor o igual
==	igual
!=	Distinto
is	idéntico objeto mismo id
is not	no idéntico objeto

2==2 True, 1==3 False, 1<3 True, 2!=2, False, 2==2.0 True, 2==2.5 False

var=0

print(var!=0) False

var=1

print(var!=o) True

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Condicionales

```
if <condicion_de_verdad>:
    <hacer esto>


if <condicion_es_verdadera>:
    <hacer esto>
else:
    <hacer_esto_si_la_condicion_es_falsa>


if <condicion_es_verdadera>:
    <hacer esto>
else:
    if <condicion_es_verdadera>:
        <hacer esto>
    else:
        <hacer_esto_si_la_condicion_es_falsa>
```

Contracción

```
if <condicion_es_verdadera>:
    <hacer esto>
elif <condicion_es_verdadera>:
    <hacer esto>
else:
        <hacer_esto_si_la_condicion_es_falsa>
```

Consideraciones

- No se debe usar un else sin un if.
- else, siempre es el fin de una cascada.
- else, sigue siendo opcional en la cascada.

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

También podremos usar el condicional llamado match, que nos permite manejar los if anidados

```
match <objeto>:
    case <1>:
        #bloque 1
    case <2>:
        #bloque 2
    case <3|4>:
        #bloque 3 o 4
    case _:
        #caso contrario ejecutar este bloque
```

ejemplo

```
match error:
    case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "soy otro error"
      case 401 | 403 | 404:
            return "Not allowed"
        case _:
            return "Algo anda mal con Internet"
```

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Iteraciones, ciclos y bucles

Un ciclo, en Python nos permite repetir una secuencia de instrucciones mientras una condición sea verdadera

Mientras que **while,** repite la ejecución siempre que la condición sea verdadera (True), si la condición es falsa (False) el cuerpo del bloque while no se ejecutara

```
while <expresion_condicional> :
        instruccion/es
```

Como se puede observar, la delimitación de bloque se realiza como en los condicionales if y else. Luego de la palabra reservada while, van la/s condición/es luego los dos puntos y las instrucciones que pertenecen, llevan una sangria o hasta cuatro espacios.

Ejemplo:

```
positivos=0
negativos=0
num=int(input("Ingrese numero: "))
while num != 0:
    if num > 0 :
        positivos=positivos+1
    else:
        negativos=negativos+1
    num=int(input("Ingrese numero: "))
print(f"Numeros positivos {positivos}")
print(F"Numeros negativos {negativos}")
```

Variable que controla el Bucle

Expresión booleana que permite ingresar al bucle mientras sea verdadera y repetir las sentencias del cuerpo del ciclo.

Actualización de la variable que controla el bucle. Fundamental para que no se generen bucles infinitos

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Ciclos for

Python nos ofrece otro tipo de ciclo, destinado a recorrer grandes colecciones de datos, tema que veremos más adelante. Veamos un ejemplo y conozcamos sus partes.

```
palabra="AIPython P1"
for letra in palabra :
print (letra)
```

Se aplican las mismas características para definir el cuerpo del ciclo mediante la sangría o tabulación, luego de los dos puntos.

Rompiendo bucles, cláusulas break y continue

Cuando necesitamos terminar el ciclo de forma inmediata, usaremos la sentencia break, la cual rompe la iteración y continua con las sentencias que están luego del ciclo.

Cuando necesitemos saltar un ciclo dentro de la iteración usaremos continue, que a diferencia de break no sale del bucle, sino que continua al siguiente turno y vuelve a la expresión de condición de inmediato.

Función range

```
    range (<inicio>, <fin>, <paso>)

for num in range (10):
    print (num)

for num in range (0, 10):
    print(num)

for num in range (0, 10, 2):
    print(num)
for num in range (10, 0,-1):
    print(num)
```

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

```
for i in range(5):
    for j in range(i+1):
        print('*', end='')
    print()
```

Un **bucle anidado** es un bucle que se encuentra dentro de otro bucle. En otras palabras, es un bucle que se ejecuta dentro de las instrucciones de otro bucle.

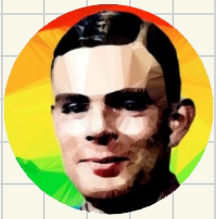
El código proporcionado utiliza dos bucles anidados para imprimir un patrón de triángulo rectángulo usando asteriscos (*). Veamos el código:

Bucle exterior (i):

- for i in range(5): - Este bucle se repite cinco veces, asignando a la variable i un valor de 0 a 4 (excluyendo 5).
- **Bucle interior (j):**
- for j in range(i + 1): - Este bucle se repite en función del valor de i en el bucle exterior.
  - En la primera iteración (i = 0), j se repite 1 veces (0 + 1).
  - En la segunda iteración (i = 1), j se repite 2 vez (1 + 1).
  - Este patrón continúa hasta la quinta iteración (i = 4), donde j se repite 5 veces (4 + 1).

Lic. Franco Herrera

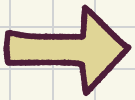
Una Ambiciosa Introducción a Python
Parte II



Operador morsa :=

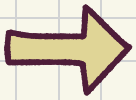
Fue introducido en la versión 3.8 y definido en la pep 572 y se llama “operador de asignación en expresión” (walrus operator). Permite la asignación de una variable dentro de una expresión. En ciertos casos mejora la eficiencia y la legibilidad del código, ya que evita la repetición de cálculos o llamadas a funciones.

```
n = len([1, 2, 3, 4])
if n > 3:
    print(f"Lista suficientemente larga ({n} elementos)")
```



```
if (n := len([1, 2, 3, 4])) > 3:
    print(f"Lista suficientemente larga ({n} elementos)")
```

```
linea = input("Ingrese texto: ")
while linea != "salir":
    print(f"Escribiste: {linea}")
    linea = input("Ingrese texto: ")
```



```
while (linea:=input("Ingrese texto: "))  != "salir":
    print(f"Escribiste: {linea}")
```

para su uso, **Siempre tener presente no reducir la claridad del código**

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

La estructura de datos listas son otro tipo de secuencias que nos permiten almacenar, objetos de tipo escalar o incluso otras colecciones o secuencias u objetos.

Algunas características

- Almacena multiples objetos
- Es ordenada
- Es mutable
- Es indexada
- Se pueden anidar
- Son iterables

Creación

- nombres=[]
- nombres=list()

Podemos darle valores iniciales:

- nombres=[‘Facundo’, ‘Laura’,’Fabiana’]
- nombres=list([‘Facundo’, ‘Laura’, ‘Fabiana’])

Añadir elemento/s al final de la lista

- lista.append(valor)

Añadir elemento/s en un indice especifico, no solo al final.

- lista.insert(ubicación, valor)

Indexación

El valor dentro del par de corchetes [valor] se denomina indice.

La operación de seleccionar elementos se llama indexación

Algo para destacar es que una expresión también puede ser un indice.

Función len

Su nombre proviene de length - longitud y nos permite obtener la dimension de la secuencia, debido a las operaciones que podemos realizar con las listas y estas al ser mutables su tamaño puede cambiar.

len(lista)

Eliminación, usamos la sentencia del

lista [4,5,6,7,-1]

del lista[3]

lista[4,5,6,-1]

print(lista[3])

-1

del lista

print (lista) nos dará un error, ya que lista ya no esta definida.

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Métodos de listas vistos

- append
- insert
- index
- remove
- reverse
- count

```
Método remove

web=['HTML','CSS','JS','Node', 'MongDB']

web.remove('Node')

web.remove('AIPYTHON ') #error el elemento no pertenece a la lista

Método pop

primer_elemento=web.pop(0)

web=['CSS','JS', 'MongDB']

web.pop() #elimina y devuelve el ultimo elemento sino pasamos un indice

web=['CSS','JS']
```

```
Método index,

web=['HTML','CSS','JS','Node', 'MongDB']

posicion=web.index('JS')

print(posicion)#2

ultimo_indice=len(web)-1

print(web[ultimo_indice])
```

Rebanadas

Una rodaja crea una nueva lista en destino, tomando elementos de la lista de origen, desde el inicio hasta el final menos uno.

lista[inicio: fin]

Copiar la lista entonces se reduce a :

lista=[5,8,6,-1]
lista_copia=lista[:], copia el contenido de la lista, no el nombre de la lista.

lista_nueva=lista[1:3]
print(lista_nueva)# [8,6]

Se pueden usar indices negativos, si omitimos el inicio de la rebanada, se tomará el 0 como tal.

lista[:fin] es igual a lista[0:fin]
lista[inicio:len(lista)]#también podemos usar expresiones y/o funciones

La instrucción del, se puede combinar con las rebanadas, veamos un ejemplos:

lista=[8,5,-3,1,11]
del lista[1:3]
print(lista) # [5,3]

```
Rebanadas en listas

web=['HTML','CSS','JS','Node', 'MongDB']

sub_web=web[:3]

print(sub_web)

ultimo_elemento=web[-1]
```

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

Existen muchas operaciones comunes al trabajar con listas numéricas como por ejemplo :

- Obtener el total de los valores almacenados
- Obtener el mínimo valor
- Obtener el máximo valor

Esto lo podemos lograr iterando sobre la secuencia como por ejemplo la suma de los elementos

```
num = [1,2,3,4,5]
total = 0
for numero in num :
    total = total + numero
```

Donde dada la lista num, primero inicializamos la variable total en cero luego iteramos sobre la colección y acumulamos la suma en total, finalemnte cuando el bucle finalice tendremos la suma de todos los valores en total.

Python nos proporciona otra forma de lograrlo a través de las funciones integradas

- sum()
- min()
- max()

```
sum([1,2,3,4,5])
```

Estas funciones toman una lista como argumento y en el caso de SUM devuelve la suma total de todos los elementos.

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

La comprensión de listas (list comprehension) es una forma compacta de crear listas mediante una única línea de código. Generalmente se usa para transformar una colección de datos en otra, aplicando una expresión o una función a cada elemento de una secuencia (como una lista, un rango, etc.) y opcionalmente filtrando elementos basados en una condición.

La sintaxis básica de una comprensión de listas es:

[expresión for elemento in iterable]

- expresión: Operación que queremos aplicar a cada elemento.
- elemento: Variable que representa cada elemento en el iterable.
- iterable: Colección de elementos que estamos iterando (lista, tupla, rango, etc.).

Ejemplos

1. Crear una lista de los cuadrados del 1 al 10

- Usando un bucle:

```
cuadrados = []
for x in range(1, 11):
    cuadrados.append(x**2)
```

- Usando comprensión de listas:

```
cuadrados = [x**2 for x in range(1, 11)]
```

La comprensión de listas también permite incluir una cláusula if para filtrar elementos.

2. Crear una lista de los números pares del 1 al 10

- Usando un bucle for tradicional con una condición:

```
pares = []
for x in range(1, 11):
    if x % 2 == 0:
        pares.append(x)
```

- Usando comprensión de listas con una condición:

```
pares = [x for x in range(1, 11) if x % 2 == 0]
```

3. Convertir una lista de palabras a mayúsculas

```
palabras = ["hola", "mundo", "python", "es", "genial"]
mayusculas = [palabra.upper() for palabra in palabras]
```

Lic. Franco Herrera