Universidad Nacional Mayor de San Marcos
Universidad del Perú, Decana de América
FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

PROGRAMA: ESPECIALIZACION EN PYTHON
MÓDULO BÁSICO

Clase 02

1.
Entrada y salidas (E/S)

Salidas en Python

La función **print()** es sin duda una de las instrucciones más usadas ya que nos permite mostrar información por consola como mensajes, números o valores de una variable. Para su uso solo le pasamos en los argumentos lo que deseamos mostrar en consola.

```
1   #   Mostramos un texto con print()
2
3   print("Entradas y Salidas con Python")
4   print("Tenemos una cadena de caracteres")
5   print("Sesión #02 Pythonistas")
6
```

```
Entradas y Salidas con Python

Tenemos una cadena de caracteres

Sesión #02 Pythonistas
```

Entradas en Python

La función **input()** permite obtener información en la terminal desde el teclado, al momento de ejecutarse esta línea en la consola esperara que ingresemos el texto que necesitemos y demos un enter para continuar y guardar el valor capturado en una variable del programa.

```
1   numero_celular = input("Ingrese su número de celular: ")
2
3   print("Su número de celuar es: ", numero_celular)
4
```

```
Ingrese su número de celular: 998342567
Su número de celuar es:  998342567

Process finished with exit code 0
```

2.

Estructuras de control de flujo

1) Asignación múltiple

Otra gran ventaja que nos provee Python es que podemos asignar en una sola instrucción múltiples variables como veremos en el siguiente ejemplo:

```
1
2   var_1, var_2, var_3 = "Primera asignación", 30, True
3
4   print("Valor de primera variable: ", var_1)
5
6   print("Valor de segunda variable: ", var_2)
7
8   print("Valor de tercera variable: ", var_3)
9
```

```
Valor de primera variable:  Primera asignaciόm
Valor de segunda variable:  30
Valor de tercera variable:  True
```

La asignación de variables múltiples, también la podemos realizar utilizando los valores dentro de una **tupla** o una **lista.**

Tuplas:

```
1
2   variable_lista = ("El atentado a las Torres Gemelas", "11 de Setiembre 2001")
3
4   suceso, fecha = variable_lista
5
6   print("Suceso:", suceso)
7
8   print("\nFecha de suceso: ", fecha)
9
```

```
Suceso: El atentado a las Torres Gemelas

Fecha de suceso:  11 de Setiembre 2001

Process finished with exit code 0
```

Listas:

```
1
2   variable_lista = ["Descubrimiento de América", "12 de Octubre de 1492"]
3
4   suceso, fecha = variable_lista
5
6   print("Suceso:", suceso)
7
8   print("\nFecha de suceso: ", fecha)
9
```

```
Suceso: Descubrimiento de América

Fecha de suceso:  12 de Octubre de 1492
```

2) Estructura de control de flujo condicionales

Las estructuras de control de flujo condicionales, son las que nos permiten evaluar si una o más condiciones se cumplen, para decir que acción se va a tomar.

Al momento de evaluar dos condiciones sólo nos puede arrojar uno de dos resultados:

verdadero o falso (***true*** o ***false***).

Operadores relaciones:

Para comparación

Símbolo	Significado	Ejemplo	Resultado
`==`	Igual que	`5 == 7`	`False`
`!=`	Distinto que	`rojo != verde`	`True`
`<`	Menor que	`8 < 12`	`True`
`>`	Mayor que	`12 > 7`	`True`
`<=`	Menor o igual que	`12 <= 12`	`True`
`>=`	Mayor o igual que	`4 >= 5`	`False`

- **Operadores lógicos:** Para evaluar más de una condición simultáneamente.

Operador	Ejemplo	Explicación	Resultado
`and`	`5 == 7 and 7 < 12`	`False and False`	`False`
`and`	`9 < 12 and 12 > 7`	`True and True`	`True`
`and`	`9 < 12 and 12 > 15`	`True and False`	`False`
`or`	`12 == 12 or 15 < 7`	`True or False`	`True`
`or`	`7 > 5 or 9 < 12`	`True or True`	`True`
`xor`	`4 == 4 xor 9 > 3`	`True o True`	`False`
`xor`	`4 == 4 xor 9 < 3`	`True o False`	`True`

2) Estructuras de control iterativas

Las estructuras de control iterativas (cíclicas o bucles) nos permite ejecutar nuestro código, muchas veces, mientras se cumpla cierta condición.

a. Bucle while:

Este bucle se ejecuta “mientras qué” se cumpla una determinada condición en pocas palabras mientras sea un ***True***.

```
1.  while condición:
2.      bloque de código
```

b. Bucle for:

El bucle **for** nos permite iterar sobre una **lista** o una **tupla o un diccionario**.

```
for <elem> in <iterable>:
    <Tu código>
```

```
ingenierias = ["Software", "Sistemas", "Industrial", "Química", "Mecánica", "Mecatrónica"]

for ingenieria in ingenierias:
    print("Ingeniería {}".format(ingenieria))
```

```
Ingeniería Software
Ingeniería Sistemas
Ingeniería Industrial
Ingeniería Química
Ingeniería Mecánica
Ingeniería Mecatrónica
```

3.
Manipulación de cadenas

Strings en Python

Los cadenas (o strings) son un tipo de datos compuestos por secuencias de caracteres que representan texto. Estas cadenas de texto son de tipo **str** y se **delimitan** mediante el uso de **comillas simples o dobles**.

Número de caracteres en mi variable str: ***len(cadena)***

Carcateréstes	P	y	t	h	o	n
Indice	0	1	2	3	4	5
Indice inverso	-6	-5	-4	-3	-2	-1

Corte de String o rebanar Strings:

Podemos seleccionar una porción de toda nuestra cadena. Para ellos usamos la siguiente denotación ***[inicio:fin:paso]***

```
>>> cadena = "Programa en Python"
>>> cadena[0:8:1]
'Programa'
>>> cadena[12:18:1]
'Python'
```

* De omitir el **paso** entender que su valor es ***1***

Operaciones con Strings:

Concatenación: Consiste en unir distintas cadenas con el signo **“+"**

```
>>> nombre = "Luke"
>>> apellido = "Skywalker"
>>> nombre + " " + apellido
'Luke Skywalker'
```

Multiplicar Strings:

```
>>> cadena = 'Python.'
>>> cadena * 4
'Python.Python.Python.Python.'
```

* Para realizar copias de cadenas usamos el signo (**)

Métodos de los Strings:

Las cadenas vienen siendo objetos los cuáles tienen diferentes métodos para facilitar su manipulación como capitalización de caracteres, remover espacios en blanco, separar palabras, etc

x.upper(): Devuelve toda la cadena en mayúsculas.

x.lower(): Devuelve toda la cadena en minúsculas.

x.title(): Devuelve la cadena en formato título.

x.replace(viejo, nuevo): Devuelve una copia a la cual se ha cambiado un carácter por otro en nuevo

x.lstrip(): Devuelve una cadena eliminando espacios en blanco al inicio.

x.rstrip(): Se comporta de la misma manera eliminando los espacios en blanco del final.

x.split(sep = None):

Devuelve una lista con los **string** que están siendo separados dentro del valor de **sep.**
En caso no se especifique el valor **sep** la cadena será separada por sus **espacios en blanco**.

```
>>> x = "Programa en Python"
>>> x.split()
['Programa', 'en', 'Python']
```

Formatear Strings:

De forma alterna tenemos el uso del método **format().**

Este método nos devuelve una copia de la cadena en donde se han sustituidos caracteres por llaves ***({})*** por los argumentos del método.

```
>>> "Mi {} se llama {} y tiene {} años".format("perro", "Ashley", "tres")
'Mi perro se llama Ashley y tiene tres años'
```

Otro modo de formatear o mostrar Strings:

Otro modo de formatear es referenciarlo por los valores de sus argumentos, por sus nombres o posición.

```
 >>> "Mi {animal} se llama {nombre} y tiene {edad} años".format(animal="perro",
nombre="Ashley", edad="tres")
 'Mi perro se llama Ashley y tiene tres años'
```

Encontrar un sub Strings dentro de una cadena de caracteres:

Se puede buscar una sub-cadena dentro de una cadena de caracteres utilizando el método **find(**cadena**)** el cual nos devolverá el índice del inicio de la cadena.

```
1
2   mensaje = "Nuevas estrategias de programación"
3   indice = mensaje.find("estrategias")
4
5   print("Indice inicial de la palabra encontada:", indice)
6
```

```
Indice inicial de la palabra encontada: 7

Process finished with exit code 0
```

4.
Funciones

Introducción a la programación funcional

Funciones en Python

Trabajar con funciones en Python nos da la gran ventaja que estas pueden ser asignadas a variables, ya que podremos saber que funciones son usadas como argumento de otras funciones y que funciones retornan funciones o ciertos valores.

```python

var_1 = 60
var_2 = 40


1 usage  new *
def sumar(a, b):
    return a + b


resultado = sumar(var_1, var_2)

print("El resultado es: {}".format(resultado))

```

```
El resultado es: 100

Process finished with exit code 0
```

Orden de parámetros:

Sabemos que cuando enviamos parámetros a una función, entendemos que el primer valor siempre va en el primer parámetro y el segundo valor al segundo parámetro, pero gracias a Python podemos cambiar esto del siguiente modo:

```
1
2   #   Priorización del valor de las variables en las funciones
3
4   def sumar(number1, number2=20):
5       return number1 + number2
6
7
8   print(sumar(10, 15))
9   print(sumar(10))
10
```

```
Resultado usando ambos parámetros:  25
Resultado usando el valor de un parámetro:  30

Process finished with exit code 0
```

Universidad Nacional Mayor de San Marcos