LENGUAJE DE PROGRAMACIÓN I

REFLEXIÓN

"La juventud solo viene una vez en la vida".

Henry Wadsworth Longfellow

TEMARIO

- Funciones lambda.
- Introducción a módulos.
- Importación absoluta y relativa.
- ¿Qué es un módulo?
- ¿Cómo importar un módulo?
- La palabra clave as.
- Módulos Incorporados.
- Paquetes.

LENGUAJE DE PROGRAMACIÓN I

SESIÓN Nº 2

FUNCIONES LAMBDA

MAG. DELY LAZO BARREDA

FUNCIONES LAMBDA

- Una función Lambda se refiere a una **pequeña función anónima** y puesto que técnicamente carecen de nombre.
- Son funciones que pueden **definir cualquier número de parámetros pero una única expresión**. Esta expresión es evaluada y devuelta.
- Se puede usar en cualquier lugar en el que la función sea requerida.

EXPLICACIÓN - 1

```
#Función Lambda
def cuadrado(x):
    return x**2

cuadrado_l=lambda x: x**2
print(cuadrado(5))
print(cuadrado_l(5))
```

```
25
25
```

EXPLICACIÓN - 2

```
#Dada la lista a = [0, 1, -1, -2, 3, -4, 5, 6, 7], filtrar solo los números mayores a
#3 y menores a 7
a = [0, 1, -1, -2, 3, -4, 5, 6, 7]
b = list(filter(lambda x: x>3 and x<7, a ))
print(b)
```

```
[5, 6]
```

EXPLICACIÓN - 3

```
#Dada la lista a = [0, 1, -1, -2, 3, -4, 5, 6, 7], filtrar solo los mayores
#que cero haciendo uso de una función que reciba dos argumentos: una lista y
#una función lambda
def mayores_0(lista, flambda):
  for elemento in lista:
    if flambda(elemento):
      print(elemento)

a = [0,1,-1,2,3,-4,5,6,7]
mayores_0(a, lambda x: x>0)
```

```
1
2
3
5
6
7
```

EXPLICACIÓN - 4

```
#Dada la lista a = [-1, 3, -5, 6, 7], mapear solo los mayores que cero
a=[-1, 3, -5, 6, 7]
valores_mapeados=map(lambda x: x if x>0 else "",a)
print(list(valores_mapeados))
```

```
['', 3, '', 6, 7]
```

EXPLICACIÓN - 5

```
#Aplicar funciones lambda a diccionarios
diccionario_original = {'a': 1, 'b': 2, 'c': 3}
diccionario_modificado = dict(map(lambda item: (item[0], item[1]
* 2), diccionario_original.items()))
print(diccionario_modificado)

diccionario_original = {'a': 10, 'b': 25, 'c': 15, 'd': 30}
diccionario_modificafo = dict(filter(lambda item: item[1] > 20,
diccionario_original.items()))
print(diccionario_modificado)
```

```
{'a': 2, 'b': 4, 'c': 6}
{'a': 2, 'b': 4, 'c': 6}
```

EJERCICIOS PROPUESTOS FUNCIONES LAMBDA

1. Cree una función lambda que permita obtener un número al cubo.
2. Cree una función lambda que te permita obtener si es número par o impar
3. Cree una función lambda que te permita obtener si es número es múltiplo de 7.
4. Cree una función lambda que determine si un número es positivo o negativo, neutro(compara con cero).
5. Cree una función lambda que permita sumar dos números enteros X e Y.

EJERCICIOS PROPUESTOS FUNCIONES LAMBDA

6. Cree una función lambda que determine obtener "a" elevado a "b".
7. Construya una función lambda que permita obtener el valor c:

$$c = \sqrt{a^2 + b^2}$$

8. Construya una función lambda que permita ingresar 2 parámetros y calcule:

$$\frac{x^2 + 10}{y^3 + 1}$$

9. Dada la lista a = [0, 1, -1, -2, 3, -4, 5, 6, 7], filtrar solo los múltiplos de dos que sean positivos.Use función lambda.
10. Dada la lista a = [“Juan”,”Ana”, “Maria”, “Jose”], filtrar solo los nombres que inician con “J”. Hacer uso de una función lambda.

LENGUAJE DE PROGRAMACIÓN I

SESIÓN Nº 2

MÓDULOS PYTHON

MAG. DELY LAZO BARREDA

Objetivos

- Conocer los principios básicos que rigen el uso de módulos en Python.
- Aprender a diseñar módulos en Python.
- Apreciar las características y ventajas provistas por los módulos en la solución de problemas.

MÓDULOS

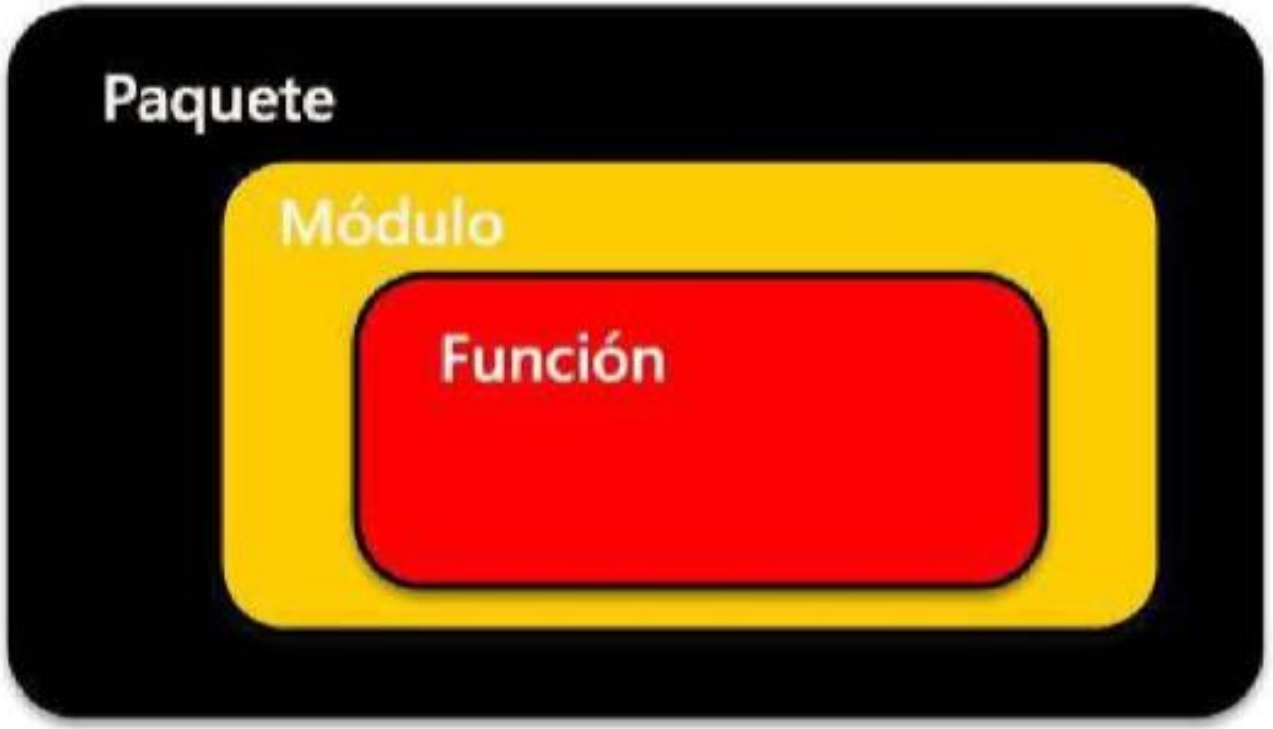
- Es un fichero .py que **alberga un conjunto de funciones, variables o clases** y que puede ser usado por otros módulos
- La declaración de importación implica dos operaciones, busca un módulo y vincula el resultado de la búsqueda al nombre en el ámbito local. **Cuando se importa un módulo, Python ejecuta todo el código en el archivo del módulo y se pone a disposición del archivo del importador.**
- Para escribir declaraciones de importación, hay algunos puntos a seguir:
  - **Las importaciones siempre deben escribirse en la parte superior del archivo**.
  - Las importaciones deben agruparse en el siguiente orden.
    - Importaciones de bibliotecas estándar(módulos integrados de Python)
    - Importaciones de terceros relacionadas.
    - Importaciones específicas de aplicaciones / bibliotecas locales

LIBRERÍAS EXTERNAS MÁS POPULARES EN PYTHON

- 1995 NumPy (Travis Oliphant, Anaconda): Arrays
- 1998 PyQt (Riverbank Computing): App Windows
- 1999 OpenCV (Gary Bradsky, INTEL): Computer Vision
- 2002 Dlib (Davis E. King): Calculo, Comp Vision, Deep Learning
- 2003 Matplotlib (John D. Hunter): Graficar
- 2005 Django (Adrian Holovaty, Simon Willison): Web
- 2007 SciKit-Learn (David Cournapeau): Machine Learning
- 2008 Pandas (Wes McKinney): Procesamiento de Datos
- 2009 Pillow (Alex Clark): Procesamiento de Graficos
- 2009 Scikit-Image (Stéfan van der Walt): Procesamiento de Graficos
- 2009 PySide (Qt Project): App Windows
- 2010 Flask (Armin Ronacher): Web
- 2012 Beautiful Soup (Leonard Richardson): Web Scrapping
- 2014 imutils (Adrian Rosebrock, PyImageSearch): Procesamiento de Graficos
- 2015 Keras (François Chollet, Google): Deep Learning
- 2015 TensorFlow (Google Brain Team): Machine Learning
- 2016 face_recognition (Adam Geitgey): Face Recognition
- 2017 PyTorch (Paszke, Gross, Chintala, Chanan, FAIR): Machine Learning
- 2018 Detectron (Girshick, Radosavovic, Gkioxari, Doll, He, FAIR): Pattern Recognition
- 2020 MediaPipe (Google): Pattern Recognition

pip install …

FUNCIÓN, MÓDULO Y PAQUETE

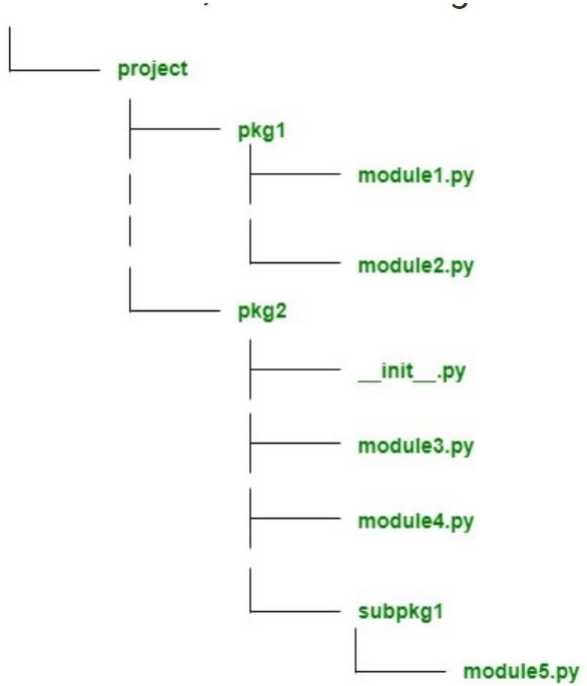


FUNCIÓN, MÓDULO Y PAQUETE

Acceso	Descripción
import Operaciones	Accediendo al paquete Operaciones
import Operaciones.OperacionesAritmeticas	Accediendo al paquete Operaciones y al modulo OperacionesAritmeticas
import Operaciones.OperacionesComparacion	Accediendo al paquete Operaciones y al modulo OperacionesComparacion
from Operaciones import OperacionesAritmeticas,OperacionesComparacion	Accediendo al paquete Operaciones y a los módulos OperacionesAritmeticas y OperacionesComparacion
from Operaciones.OperacionesAritmeticas import suma	Accediendo al paquete Operaciones y a los módulos OperacionesAritmeticas y función suma

IMPORTACIONES ABSOLUTAS

- La importación absoluta **implica la ruta completa**, es decir, desde la carpeta raíz del proyecto hasta el módulo deseado. Un estado de importación absoluto que indica que el recurso se va a importar utilizando su ruta completa desde la carpeta raíz del proyecto.



- `pkg1 / module1.py` contener una función, fun1
- `pkg2 / module3.py` contener una función, fun2
- `pkg2 / subpkg1 / module5.py` contener una función fun3

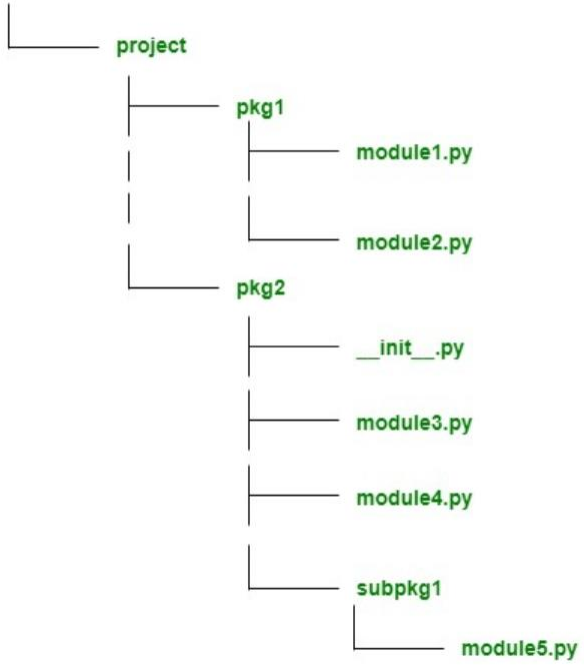
```
from pkg1.module1 import fun1
from pkg2 import module3
from pkg2.module3 import fun2
from pkg2.subpkg1.module5 import fun3
```

PROS Y CONTRAS DE LAS IMPORTACIONES ABSOLUTAS

- **Pros:**
  - Las importaciones absolutas son muy útiles porque **son claras y van al grano**.
  - La importación absoluta **es fácil de saber exactamente desde dónde está el recurso importado**, con solo mirar la declaración.
  - La importación absoluta sigue siendo válida incluso si cambia la ubicación actual de la declaración de importación.
- **Contras:**
  - Si la estructura del directorio es muy grande, el uso de importaciones absolutas no es significativo. En tal caso, el uso de importaciones relativas funciona bien.

IMPORTACIONES RELATIVAS

- La importación relativa especifica el objeto o módulo importado desde su ubicación actual, que es la ubicación donde reside la declaración de importación.



- pkg1 / module1.py contener una función, fun1
- pkg2 / module3.py contener una función, fun2
- pkg2 / subpkg1 / module5.py contener una función fun3

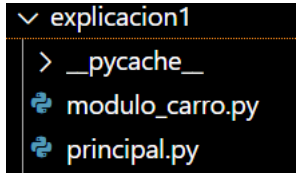
```
from .module1 import fun1
from .module3 import fun2
from .subpackage1.module5 import fun3
```

```
from .moduleY import spam
from .moduleY import spam as ham
from . import moduleY
from ..subpackage1 import moduleY
from ..subpackage2.moduleZ import eggs
from ..moduleA import foo
from ...package import bar
from ...sys import path
```

PROS Y CONTRAS DE LAS IMPORTACIONES RELATIVAS

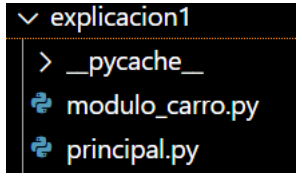
- **Pros:**
  - Trabajar con importaciones relativas es conciso y claro.
  - Según la ubicación actual, **reduce la complejidad de una declaración de importación**.
- **Contras:**
  - Las importaciones relativas no son tan legibles como las absolutas.
  - Usar importaciones relativas no es fácil porque es muy difícil saber la ubicación de un módulo.

EXPLICACIÓN - 1




```
#Dado el paquete explicación1 crearemos el módulo modulo_carro y el módulo principal
#modulo_carro.py
def encenderCarro():
    return "El carro está encendido ..."

datos={
    "placa":"F4S268",
    "modelo":"CVR",
    "marca":"Honda",
    "color":"plomo",
    "anio":2023
}
```

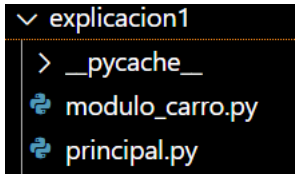
EXPLICACIÓN - 1




```
#modulo_principal.py
import modulo_carro
print(modulo_carro.encenderCarro())
```

```
El carro está encendido ...
```

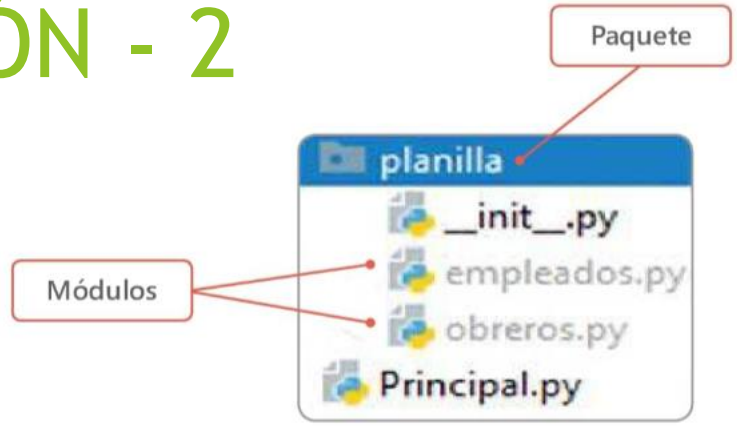
EXPLICACIÓN - 1




```
#modulo_principal.py (otra alternativa)
from modulo_carro import encenderCarro
from modulo_carro import datos
print(encenderCarro())

print("********DATOS********")
for clave,valor in datos.items():
    print(clave,valor)

print("********DATOS********")

print(datos["placa"])
print(datos["modelo"])
print(datos["marca"])
print(datos["color"])
print(datos["anio"])
```

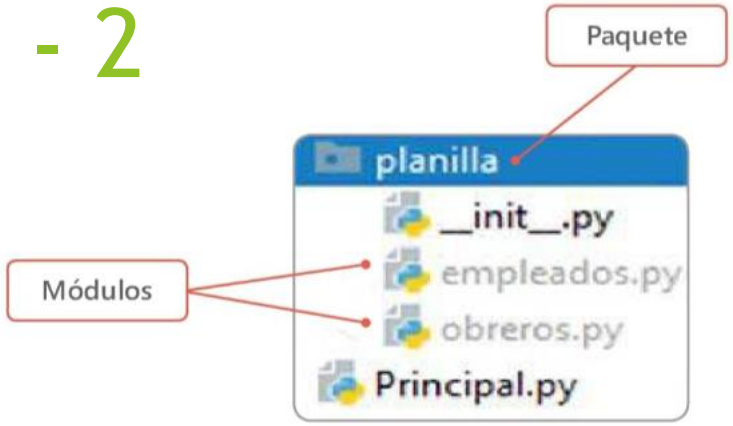
```
El carro está encendido ...
********DATOS********
placa F4S268
modelo CVR
marca Honda
color plomo
anio 2023
********DATOS********
F4S268
CVR
Honda
plomo
2023
```

EXPLICACIÓN - 2



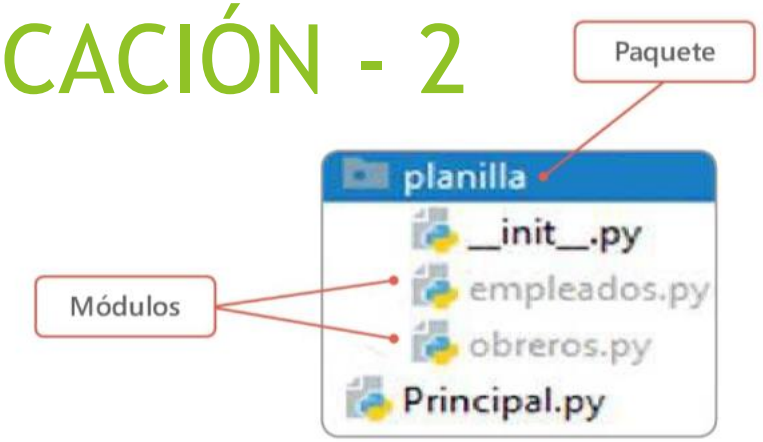

```
#empleados.py
def datos(nombres,apellidos,cargo,horasm,sueldo):
    print("Su Nombre es :"+nombres)
    print("Su Apellidos son :"+apellidos)
    print("Su cargo es:"+cargo)
    print("Su Horas Mensuales Trabajadas son :"+horasm)
    print("Su Sueldo es :"+sueldo)

def pagoMensual(horasm,sueldo):
    pagoxhora=sueldo/30/8
    pago=pagoxhora*horasm
    print("Su pago mensual es :"          +str(pago))
```

EXPLICACIÓN - 2




```
#obreros.py
def datos(nombres,apellidos,cargo,horasm,sueldo):
    print("Su Nombre es :"+nombres)
    print("Su Apellidos son :"+apellidos)
    print("Su cargo es:"+cargo)
    print("Su Horas Mensuales Trabajadas son :"+horasm)
    print("Su Sueldo es :"+sueldo)

def pagoSemanal(horass,salario):
    pagoxhora=salario/7/8
    pago=pagoxhora*horass
    print("Su pago semanal es:"          +str(pago))
```

EXPLICACIÓN - 2




```
Su Nombre es :Dely
Su Apellido es :Lazo
Su cargo es:Ingeniero
Su Horas Mensuales Trabajadas son :30
Su Sueldo es :8000
Su pago mensual es :1000.0000000000001
```

```
#Uso de Paquetes y Módulos
#accediendo al paquete planilla
import planilla
#accediendo al paquete planilla y al módulo empleados
import planilla.empleados
#accediendo al paquete planilla y a los módulos empleados y obreros
from planilla import empleados,obreros
#accediendo al paquete planilla y al módulo empleados y función datos
from planilla.empleados import datos, pagoMensual

datos('Dely','Lazo','Ingeniero','30','8000')
pagoMensual(30,8000)
```

EJEMPLO 1

- Crear una carpeta Lab02 y abrirla VS Code.
- Crear una subcarpeta modulo_0 y dentro de ella dos archivos: saludos.py y principal.py

```
# saludos.py
def saludar():
    print('Bienvenido al curso LP1 ... desde el módulo saludos')
def saludar_personalizado(nombre):
    print('Bienvenido ', str(nombre), ' al curso Lenguaje de Programación I’)

#principal.py
import saludos
saludos.saludar()
print('================================================')
saludos.saludar_personalizado('Dely')
```

```
Bienvenido al curso LP1 ... desde el módulo saludos
================================================
Bienvenido  Dely  al curso Lenguaje de Programación I
```

EJEMPLO 1

- Crear los archivos principal1.py y principal2.py

```
from saludos import saludar,saludar_personalizado
saludar()
print('==================================================')
saludar_personalizado('Dely')

# principal2.py
from saludos import *
saludar()
print('==================================================')
saludar_personalizado('Dely')
```

```
Bienvenido al curso LP1 ... desde el módulo saludos
==================================================
Bienvenido  Dely  al curso Lenguaje de Programación I
```

EJEMPLO 2

- Crear una subcarpeta modulo_1 y dentro de ella el archivo: operaciones.py.
- Crear el archivo principal.py en la carpeta Lab02

```
#operaciones.py
def suma(num01, num02):
    return num01 + num02

TEXTO="Esto es mi constante"
variable=0
```

```
#principal.py
#Importar Módulo Operaciones Modalidad 01

import modulo_1.operaciones as modulo

numero01 = int(input("Ingrese primer número: "))
numero02 = int(input("Ingrese segundo número: "))

resultado_suma = modulo.suma(numero01, numero02)
print(f"El resultado de la suma es {resultado_suma}")
```

EJEMPLO 2

- Crear una subcarpeta modulo_1 y dentro de ella el archivo: operaciones.py.
- Crear el archivo principal.py en la carpeta Lab02

```
#operaciones.py
def suma(num01, num02):
    return num01 + num02

TEXTO="Esto es mi constante"
variable=0
```

```
#principal.py
#Importar Módulo Operaciones Modalidad 02

from modulo_1.operaciones import suma

numero01 = int(input("Ingrese primer número: "))
numero02 = int(input("Ingrese segundo número: "))

resultado_suma = suma(numero01, numero02)
print(f"El resultado de la suma es {resultado_suma}")
```

EJEMPLO 2

- Crear una subcarpeta modulo_1 y dentro de ella el archivo: operaciones.py.
- Crear el archivo principal.py en la carpeta Lab02

```
#operaciones.py
def suma(num01, num02):
    return num01 + num02

TEXTO="Esto es mi constante"
variable=0
```

```
#principal.py
#Importar Módulo Operaciones Modalidad 03

from modulo_1.operaciones import *

numero01 = int(input("Ingrese primer número: "))
numero02 = int(input("Ingrese segundo número: "))

resultado_suma = suma(numero01, numero02)
print(f"El resultado de la suma es {resultado_suma}")
```

EJEMPLO 2

- Crear una subcarpeta modulo_1 y dentro de ella el archivo: operaciones.py.
- Crear el archivo principal.py en la carpeta Lab02

```
#operaciones.py
def suma(num01, num02):
    return num01 + num02

TEXTO="Esto es mi constante"
variable=0
```

```
#principal.py
#Llamada a una constante
import modulo_1.operaciones as modulo

print(modulo.TEXTO)

#Llamada a una variable
print(modulo.variable)
```

EJEMPLO 3: USO DEL MÓDULO MATH

```
import math
print(math.pi) #Pi, 3.14...
print(math.e) #Número de Euler, 2.71...
print(math.degrees(2)) #2 radianes = 114.59 grados
print(math.radians(60)) #60 grados = 1.04 radianes
print(math.sin(2)) #Seno de 2 radianes
print(math.cos(0.5)) #Coseno de 0.5 radianes
print(math.tan(0.23)) #Tangente de 0.23 radianes
print(math.factorial(5)) #1 * 2 * 3 * 4 * 5 = 120
print(math.sqrt(49)) #Raíz cuadrada de 49 = 7
```

EJEMPLO 4: USO DEL MÓDULO RANDOM

```
import random
# Flotante aleatorio >= 0 y < 1.0
print(random.random())
# Flotante aleatorio >= 1 y <10.0
print(random.uniform(1,10))
# Entero aleatorio de 0 a 9, 10 excluído
print(random.randrange(10))
# Entero aleatorio de 0 a 100
print(random.randrange(0,101))
# Entero aleatorio de 0 a 100 cada 2 números, múltiples de 2
print(random.randrange(0,101,2))
# Entero aleatorio de 0 a 100 cada 5 números, múltiples de 5
print(random.randrange(0,101,5))
```

EJEMPLO 4: USO DEL MÓDULO DATETIME

```
from datetime import datetime

dt = datetime.now()     # Fecha y hora actual
print(dt)
print(dt.year)          # año
print(dt.month)         # mes
print(dt.day)           # día
print(dt.hour)          # hora
print(dt.minute)        # minutos
print(dt.second)        # segundos
print(dt.microsecond)   # microsegundos
print("{}:{}:{}".format(dt.hour, dt.minute, dt.second))
print("{}/{}/{}".format(dt.day, dt.month, dt.year))
```

EJEMPLO 5: USO DEL MÓDULO OS - PATHLIB

```
import os
from pathlib import Path

#Ruta del script de python
print(os.getcwd())
print(Path.cwd())

print(type(os.getcwd()))
print(type(Path.cwd()))

#Lista las carpetas
print(os.listdir())
print(list(Path().iterdir()))
```

EJEMPLO 6: PAQUETES

- Crear una subcarpeta paquete y dentro de ella el archivo vacío __init__.py.
- Crear el archivo saludos.py en la subcarpeta paquete.

```
#saludos.py
def saludar():
    print('Bienvenido al curso LP1 ... desde el módulo saludos')

def saludar_personalizado(nombre):
    print('Bienvenido ", str(nombre), " al curso LP1')
```

EJEMPLO 6: PAQUETES

- Crear el archivo principalpaquete.py en la subcarpeta Lab02.

```
from paquete.saludos import *

saludar()
print('==================================')
saludar_personalizado('Dely')
```

EJERCICIOS PROPUESTOS

EJERCICIOS PROPUESTOS

1. Crear una carpeta propuesto1 desde VSCode, la cual este ubicada dentro de lab_02, dentro de ella crear los archivos geometria.py, aritmetica.py y validaciones.py.
   - En el módulo geometria.py considerar las siguientes funciones: area_triangulo, area_cuadrado, area_rectangulo.
   - En el módulo aritmetica.py considerar las siguientes funciones: suma, resta, multiplicacion, division, potencia y resto de dos números enteros. En las funciones del módulo deberá de haber tratamiento de errores para evitar que se quede bloqueada una funcionalidad, eso incluye la división entre cero.
   - En el módulo de validaciones.py considerar la función que solo pueden ingresar números, los cuales pasarán como argumentos en las funciones del módulo de geometría y aritmética.
   - Crear un archivo calculos.py desde el cual debe llamar a los tres módulos: geometría, aritmética y validaciones, deberá ejecutar las diversas funciones que cada uno de estos tiene.

EJERCICIOS PROPUESTOS

2. Crear una carpeta propuesto2, en la cual desarrolle un programa que permita cargar una lista con 10 valores enteros, los cuales deben ser números aleatorios entre 0 y 100. Mostrar la lista obtenida por pantalla. Dividir el programa en dos archivos funciones.py y principal.py (donde se encontrará la ejecución de las funciones).
3. Crear una carpeta propuesto3, en el debe realizar un programa que solicite un valor entero. Después mostrar por pantalla la raíz cuadrada, raíz cúbica, el valor elevado al cuadrado y al cubo de dicho número. (Utilizar el módulo math de python). Dividir el programa en dos archivos funciones_math.py y principal.py(donde se encontrará la ejecución de las funciones y deberá contar con una función que muestre las opciones de menú como: raíz cuadrada, raíz cúbica, potencia al cuadrado, potencia al cubo). Los resultados debe mostrarlos redondeado a dos decimales. Puede realizar una o cuatro funciones para lo requerido en el archivo funciones.py, controlar errores como raíz cuadrada de un valor negativo. (Usar excepciones).

EJERCICIOS PROPUESTOS

4. Crear una carpeta propuesto4, en él desarrollar un módulo para validación de contraseñas. Dicho módulo, deberá cumplir con los siguientes criterios de aceptación:
   - La contraseña debe contener un mínimo de 8 caracteres.
   - Una contraseña debe contener al menos una letra minúscula, una letra mayúscula y un número.
   - La contraseña no puede contener espacios en blanco.
   - Contraseña válida, retorna True.
   - Contraseña no válida, retorna el mensaje "La contraseña elegida no es segura".
   - Dividir el programa en dos archivos validaciones.py y principal.py. Utilice los métodos de string como: islower(), isdigit(), isupper(), isspace().
5. Crear una carpeta propuesto5 y desarrollar un programa que valide una fecha en el formato dd/mm/yyyy, use el módulo datetime. Dividir el programa en dos archivos validaciones.py y principal.py.

EJERCICIOS PROPUESTOS

6. Crear una carpeta propuesto6 y desarrollar un programa agenda con 3 opciones:
   - Opcion 1: Registrar agenda.
   - Opcion 2: Listar agenda.
   - Opcion 3: Salir.
   - ✓ Si el usuario ingresa otra opción, el sistema debe de volver a mostrar las opciones.
   - ✓ Si el usuario ingresa el valor de 1, el programa debe solicitar:
   - ✓ Registrar nombre (considerar la primera letra del nombre en mayúscula, en caso de que se ingrese en minúscula el texto y en caso el texto se encuentre vacío el programa debe de volver a solicitar).
   - ✓ Registrar dirección (considerar la primera letra en mayúscula) y número de teléfono o celular (validar que solo se ingrese números). Al finalizar el registro, este se debe de almacenar en un diccionario.
   - ✓ Una vez que el usuario termine de realizar el registro se debe de volver a mostrar las opciones.
   - ✓ Si el usuario ingresa a la opción 2 le debe de mostrar la lista con los registros ingresados. Una vez que el usuario termine de mostrar el listado se debe de volver a mostrar las opciones.
   - ✓ Si el usuario ingresa la opción 3 la aplicación debe de finalizar.
   - ✓ Dividir el programa en dos archivos validaciones.py y principal.py.

CUESTIONARIO

CUESTIONARIO

1. ¿Qué es un módulo?
2. ¿Qué es el diseño modular?
3. ¿Qué ventajas ofrece el uso de módulos?
4. ¿Para qué son útiles los módulos?
5. ¿Cómo hacemos uso de un módulo?
6. ¿De qué forma se importa un módulo?
7. ¿Cómo se hace la importación de un módulo con import?
8. ¿Cómo se hace la importación con from?
9. ¿Cuándo hacemos uso de import?
10. ¿Cuándo hacemos uso de from para la importación?

GRACIAS!