Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

alkemy

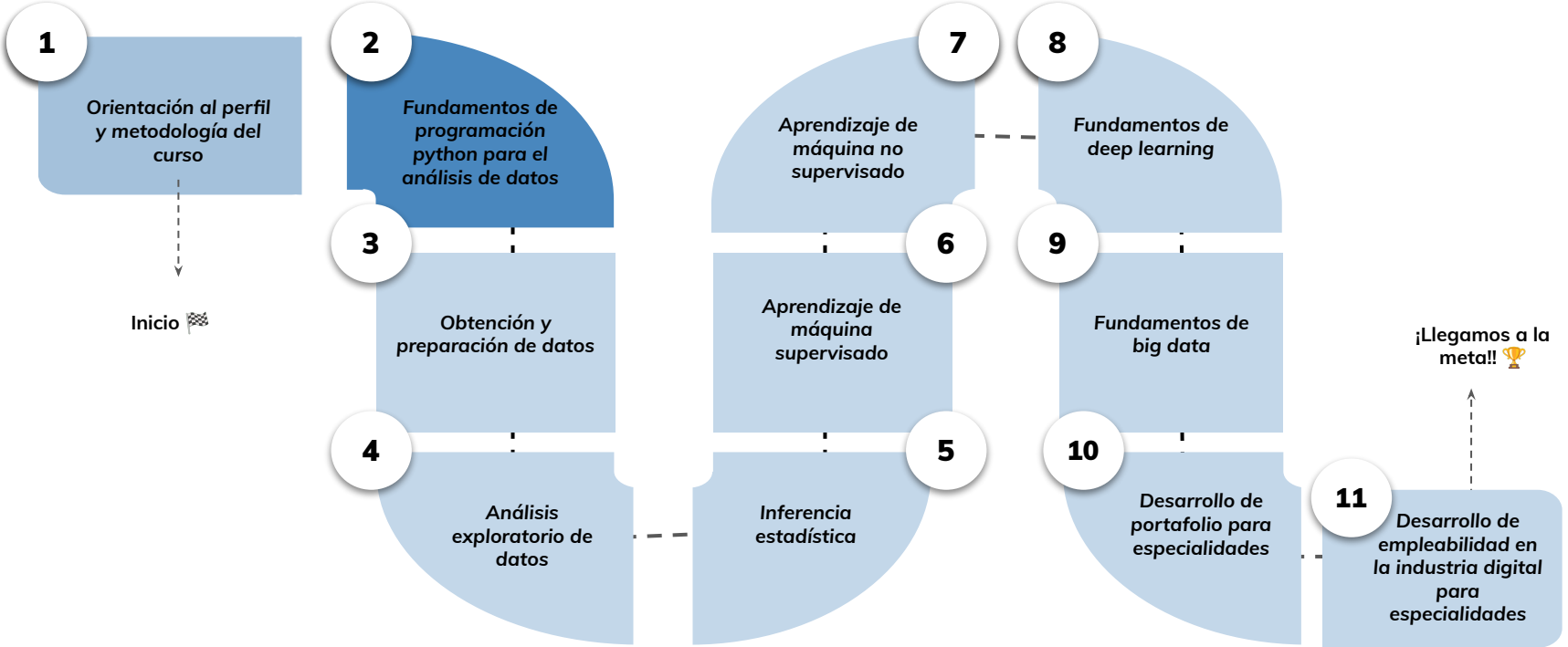
Clase grabada

Funciones y módulos- Parte 1

Aprendizaje Esperado 4: Codificar una rutina utilizando funciones preconstruidas, funciones personalizadas o de un módulo para resolver un problema de baja complejidad de acuerdo al lenguaje python.

Hoja de ruta

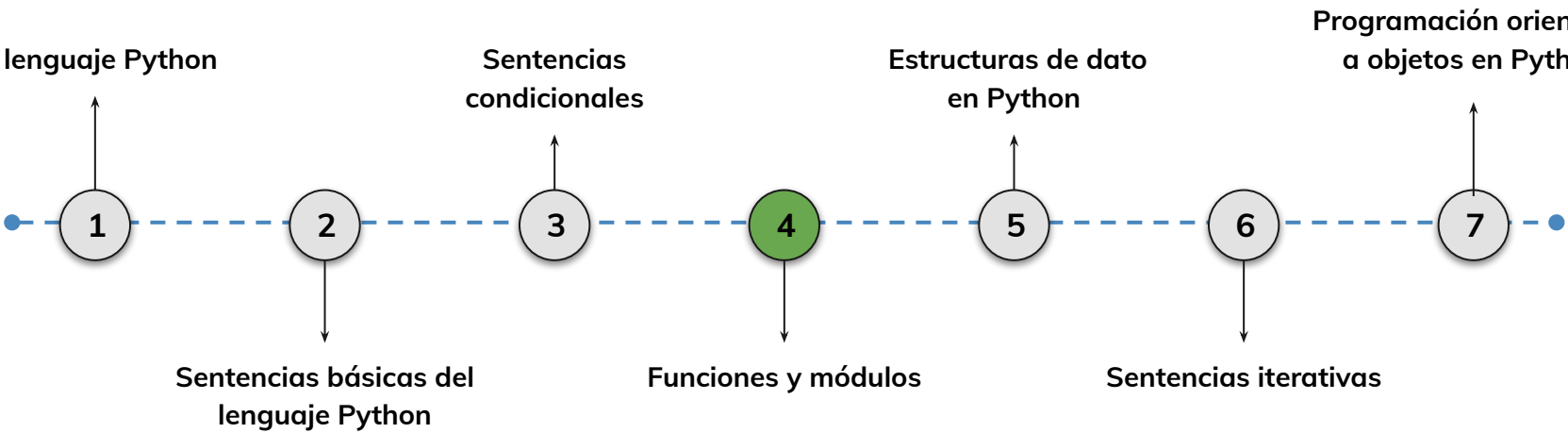
¿Cuáles skills conforman el programa? **Fundamentos de Ciencia de Datos**



alkemy

Roadmap de lecciones

*¿Cuáles **lecciones** estaremos estudiando en este módulo?*



alkemy

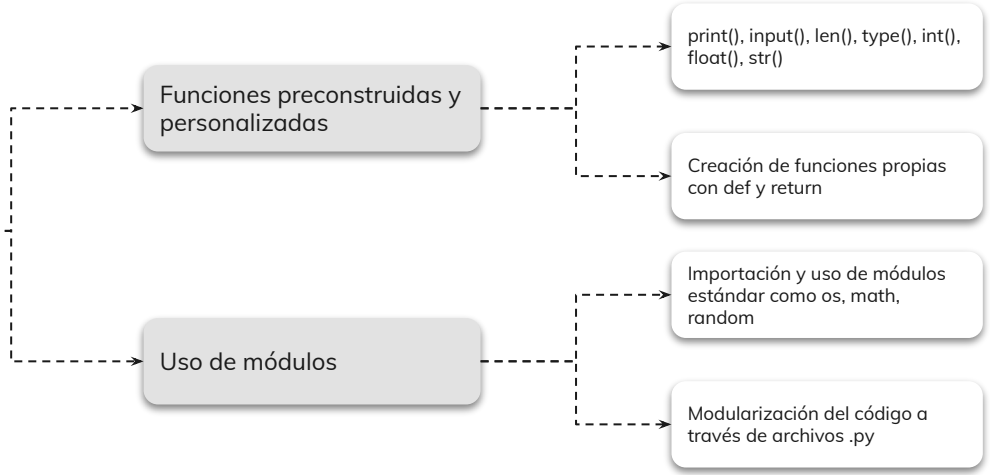
Learning Path

¿Cuáles temas trabajaremos hoy?

4.

Uso de funciones y módulos en la resolución de problemas simples con Python.

Conoceremos funciones predefinidas, personalizadas y módulos como estructuras clave para organizar y reutilizar código de manera efectiva, resolviendo desafíos cotidianos con mayor claridad y mantenimiento.



alkemy

Objetivos de aprendizaje

¿Qué aprenderás?

- Usar funciones integradas para realizar tareas básicas en Python.
- Definir y utilizar funciones propias con parámetros y retorno.
- Documentar funciones correctamente con docstrings.
- Importar y utilizar funciones desde módulos estándar de Python.
- Crear código más modular, legible y reutilizable.

Repaso clase anterior

¿Quedó alguna duda?

En la clase anterior trabajamos 📚:

- → *Profundizamos en las sentencias condicionales if-elif-else y expresiones ternarias.*
- → *Aplicamos condiciones múltiples para tomar decisiones basadas en múltiples variables.*
- → *Aprendimos a simplificar código usando estructuras lógicas bien organizadas.*
- → *Creamos una rutina que clasificaba usuarios según rol y edad usando decisiones anidadas.*

Funciones y módulos

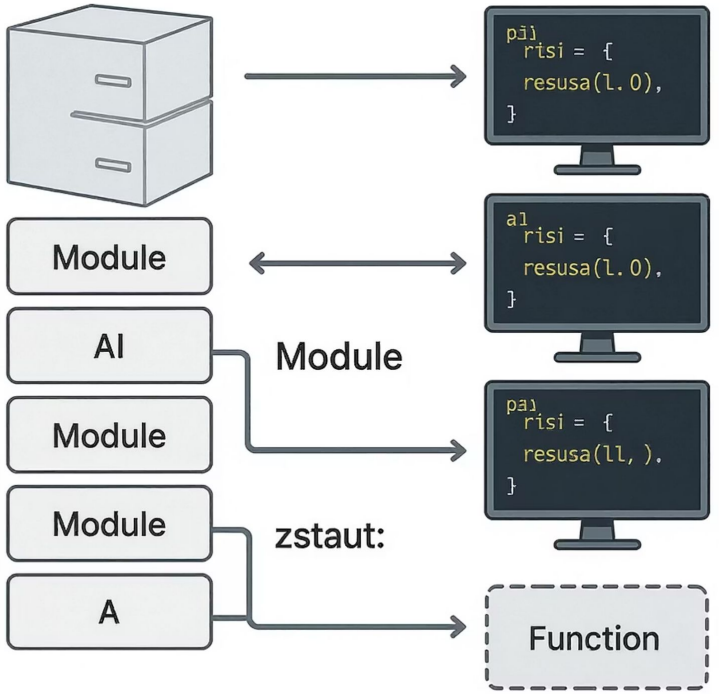
Funciones y módulos

Las funciones y módulos son herramientas fundamentales en Python que permiten estructurar, reutilizar y organizar el código de manera eficiente. Este manual aborda el concepto y uso de funciones y módulos, presentando las herramientas que Python ofrece para maximizar la eficiencia en la programación.

alkemy

Introducción

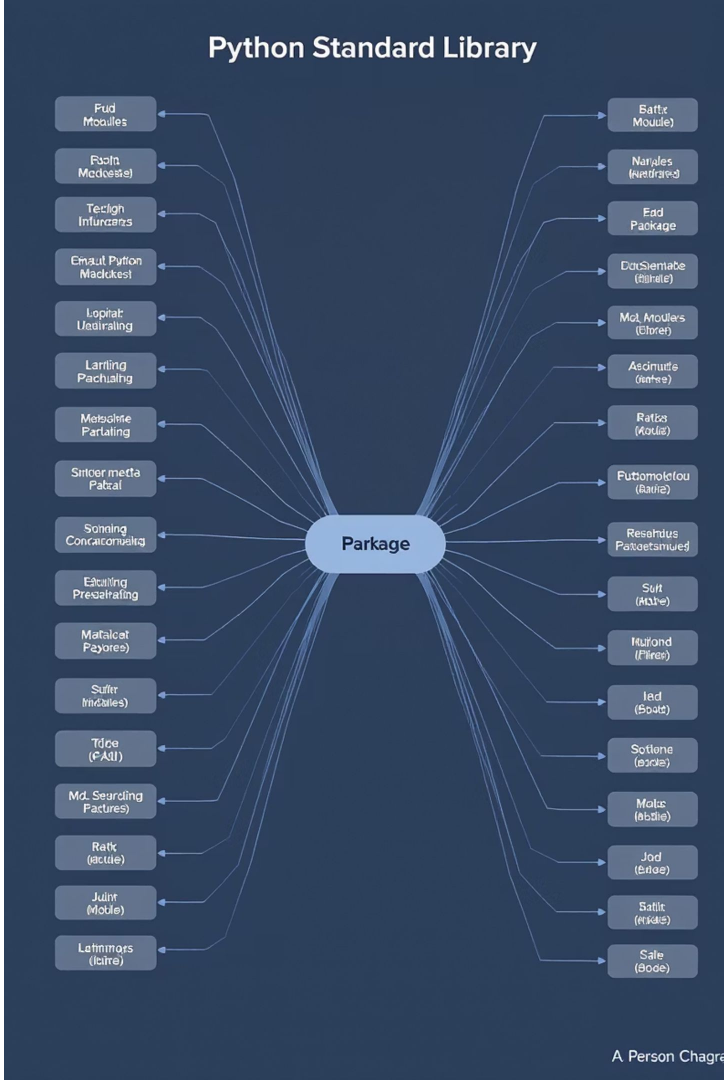
Una función es un bloque de código que realiza una tarea específica y que se puede llamar en cualquier momento, lo que facilita el desarrollo de programas estructurados y reduce la repetición de código. Los módulos, por otro lado, son archivos que contienen funciones, clases y variables, y que se pueden importar en otros programas para extender su funcionalidad sin tener que escribir todo desde cero.



alkemy

Biblioteca estándar de Python

Python, con su amplia biblioteca estándar, incluye muchas funciones y módulos pre construidos que permiten realizar operaciones comunes y especializadas sin esfuerzo adicional. Además, permite la creación de funciones personalizadas y módulos que se adaptan a las necesidades específicas de cada proyecto.



¿Qué es una función?

Concepto básico

Una función es un bloque de código que se define para realizar una tarea específica y que se puede ejecutar varias veces en diferentes partes del programa.

Definición en Python

En Python, las funciones se definen con la palabra clave def, seguida del nombre de la función, paréntesis y dos puntos.

Propósito

Las funciones ayudan a modularizar el código, dividiéndolo en fragmentos más pequeños y manejables que pueden usarse repetidamente.

Ventajas de utilizar funciones

Reutilización de código

Permiten usar el mismo código en diferentes partes del programa sin necesidad de reescribirlo.

Modularidad

Dividen un problema en tareas más pequeñas y manejables, facilitando el trabajo en equipo.

Legibilidad

Mejoran la comprensión del código al encapsular operaciones en nombres descriptivos.

Mantenimient

Facilitan la corrección de errores, ya que los cambios solo se realizan en un lugar.

Definición y uso de funciones

Para definir una función en Python, se utiliza la palabra clave def, seguida del nombre de la función, paréntesis y dos puntos. El bloque de código de la función debe estar indentado.

```
def saludo():    print("¡Hola, mundo!")
# Llamando a la función saludo()
```

Documentación de funciones

Es buena práctica documentar cada función, explicando brevemente lo que hace. En Python, se utiliza el "docstring" para añadir una descripción dentro de la función, lo cual ayuda a otros programadores (o a uno mismo en el futuro) a entender rápidamente el propósito y uso de cada función.

```
def saludo():
    """Esta función imprime un saludo de bienvenida."""
    print("¡Hola! Bienvenido al programa.")
```

El docstring se coloca entre triple comillas justo después de la definición de la función.

Funciones preconstruidas de Python

Python incluye una serie de funciones preconstruidas (o funciones integradas) que están disponibles de forma inmediata y no requieren importación de módulos adicionales. Estas funciones cubren operaciones comunes, como conversión de tipos, operaciones matemáticas básicas, manipulación de cadenas y colecciones, entre otras.

Funciones comunes

print(), len(), int(), float(), str(), input(), type(), etc.

Ventajas

Ahorran tiempo y esfuerzo, ya que permiten realizar tareas comunes de manera rápida y eficiente.

Optimización

Están optimizadas para funcionar correctamente y con buen rendimiento.

Función len()

La función len() permite conocer la longitud de una secuencia, como una lista o cadena de texto.

```
texto = "Hola, mundo"longitud =
len(texto)print(longitud)  # Imprime: 11
lista = [1, 2, 3, 4, 5]longitud =
len(lista)print(longitud)  # Imprime: 5
```

```
texto = "Python"
print(len(texto))  # Salida: 6
```

Funciones print() e input()

La función print() se utiliza para mostrar información en pantalla, mientras que input() permite capturar datos ingresados por el usuario desde la consola. Estas funciones son esenciales para interactuar con el usuario y presentar resultados.

```
nombre = input("¿Cuál es tu nombre? ")
print("Hola,", nombre)
```

Ejemplo de uso de input() y print() juntos para solicitar y mostrar información al usuario.

Función type()

La función type() permite verificar el tipo de dato de una variable, lo cual es útil para asegurar que los valores sean los esperados.

```
x = 10y = "Hola"z = [1, 2, 3]print(type(x))  #
print(type(y))  # print(type(z))  #
```

```
edad = 25
print(type(edad))  # Salida: <class 'int'>
```

Conversiones de tipo

Las funciones int(), float() y str() convierten valores de un tipo de dato a otro, lo cual es esencial en operaciones matemáticas y de manipulación de texto.

```
# Conversión de string a entero
numero_texto = "123"numero = int(numero_texto)
print(numero + 10)  # 133
# Conversión de entero a string
numero = 456
texto = str(numero)
print("El número es: " + texto)
```

```
numero = "10"
numero_int = int(numero)
print(numero_int + 5)  # Salida: 15
```

Funciones personalizadas

Las funciones personalizadas son aquellas definidas por el usuario para realizar tareas específicas que no están cubiertas por las funciones pre-construidas. Se definen utilizando la palabra clave def, seguida de un nombre de función y paréntesis. Dentro del cuerpo de la función, se coloca el código que define su comportamiento.

Ejemplo de función personalizada

```
def cuadrado(numero):    """    Calcula el cuadrado
de un número.        Args:        numero: El número
a elevar al cuadrado            Returns:        El
cuadrado del número    """    return numero ** 2#
Uso de la funciónresultado =
cuadrado(5)print(resultado)  # Imprime: 25
```

```
def cuadrado(numero):
    return numero ** 2
```

Esta función recibe un número como argumento y devuelve su cuadrado.

Parámetros y retorno de funciones

Las funciones pueden recibir parámetros, que son valores externos que se pasan a la función para que trabaje con ellos. Los parámetros se definen entre paréntesis al declarar la función. Además, las funciones pueden retornar valores usando la palabra clave return.

```python
def suma(a, b):
    return a + b

resultado = suma(5, 3)
print(resultado)  # Salida: 8
```

Ejemplo de función con parámetros y retorno que calcula el área de un rectángulo.

Funciones con parámetros predeterminados

Python permite definir valores predeterminados para los parámetros, de modo que si no se pasan ciertos argumentos, la función utilice el valor por defecto. Esto es útil para funciones que necesitan parámetros opcionales.

```
def saludo(nombre="Usuario"):
    print("Hola,", nombre)

saludo()          # Salida: Hola, Usuario
saludo("Ana")     # Salida: Hola, Ana
```

En este ejemplo, si no se proporciona un valor para el parámetro "veces", la función utilizará el valor predeterminado 1.

Ventajas de las funciones personalizadas

Soluciones específicas

Permiten crear código adaptado a las necesidades particulares de cada proyecto.

Modularidad

Mejoran la organización del código, dividiéndolo en componentes independientes.

Reutilización

Facilitan el uso del mismo código en diferentes partes del programa o en distintos proyectos.

Colaboración

Facilitan el trabajo en equipo al permitir que diferentes programadores trabajen en distintas funciones.

¿Qué es un módulo?

Un módulo es un archivo que contiene definiciones de funciones, variables y clases, que se pueden importar y usar en otros programas. Los módulos permiten organizar y dividir el código en partes más pequeñas y manejables, facilitando su reutilización en diferentes proyectos. En Python, cualquier archivo .py es un módulo que se puede importar en otros scripts.

Ejemplo de módulo

Por ejemplo, si tenemos un archivo llamado operaciones.py con la siguiente función:

```
# Archivo: operaciones.pydef sumar(a, b):    """Suma dos números y devuelve el resultado."""
return a + b
```

Podemos importar este módulo en otro archivo y usar la función sumar().

Importancia de los módulos

Reutilización

Los módulos permiten reutilizar código en diferentes proyectos, ahorrando tiempo y esfuerzo.

Organización

Facilitan la organización del código en componentes lógicos y manejables.

Colaboración

Permiten que varios programadores trabajen en diferentes módulos sin interferir entre sí.

Mantenimiento

Mejoran la mantenibilidad del código, ya que los cambios en un módulo no afectan a otros.

Modularidad y mantenibilidad

El uso de módulos mejora la modularidad del código, permitiendo que cada archivo se enfoque en una tarea específica. Esto facilita el mantenimiento, ya que cualquier cambio en la lógica de una función solo requiere actualizar el módulo correspondiente, sin afectar otras partes del programa.

Organización de funcionalidades

Módulo de operaciones

Agrupa funciones matemáticas y operaciones aritméticas.

Módulo de archivos

Contiene funciones para leer, escribir y manipular archivos.

Módulo de base de datos

Incluye funciones para conectar y consultar bases de datos.

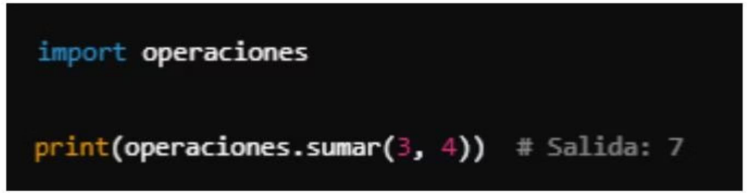
Módulo de gráficos

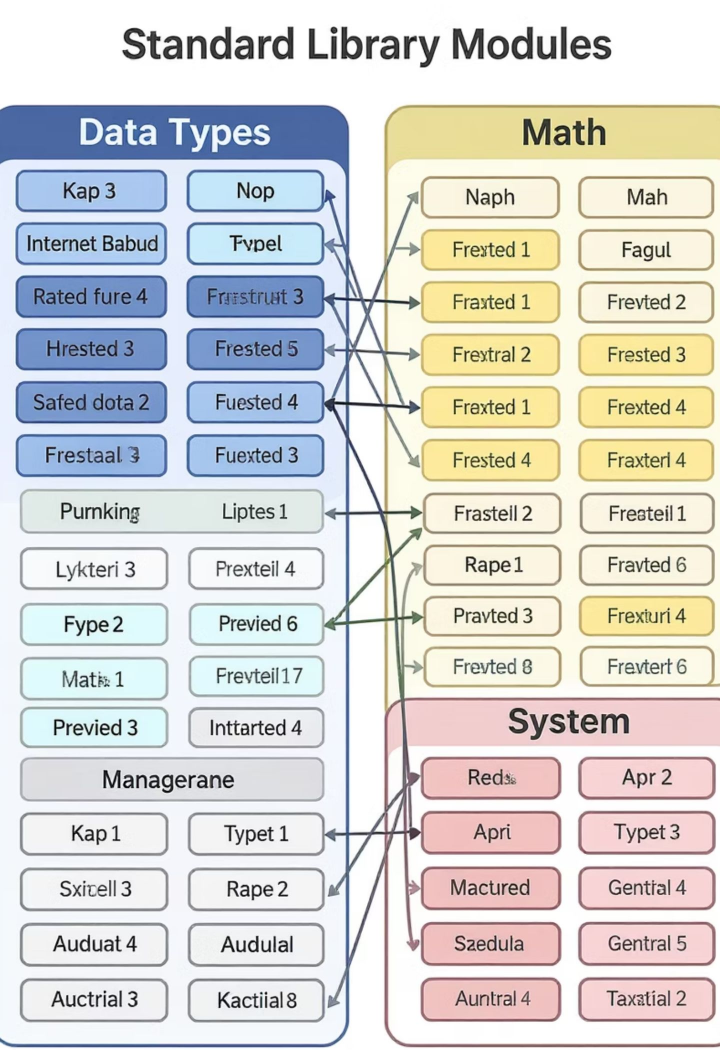
Proporciona funciones para crear y mostrar visualizaciones.

Importación de un módulo

Para importar el módulo operaciones.py, basta con usar la palabra clave import:

```
# Importar el módulo operacionesimport operaciones# Usar la función sumar del móduloresultado = operaciones.sumar(5, 3)print(resultado)  # Imprime: 8
```





La librería estándar de Python

La librería estándar de Python es una colección de módulos que vienen incluidos con Python y que proporcionan funciones y herramientas listas para usar en una gran variedad de tareas. Incluye módulos para manipulación de archivos, operaciones matemáticas, gestión de fechas, acceso a internet, y más.

alkemy

Ventajas de la librería estándar

Disponibilidad inmediata

Viene incluida con Python, sin necesidad de instalaciones adicionales.

Documentación completa

Cuenta con documentación detallada y ejemplos de uso.

Código probado

Los módulos están bien probados y optimizados para su uso.

Soluciones listas

Ofrece soluciones para tareas comunes sin necesidad de escribir código desde cero.

Módulo os para archivos

El módulo os en la librería estándar permite trabajar con el sistema de archivos.

Por ejemplo:

```
import os# Listar archivos en el directorio
actualarchivos = os.listdir('.')print(archivos)#
Verificar si un archivo existeexiste =
os.path.exists('archivo.txt')print(existe)# Crear un
directorioos.mkdir('nuevo_directorio')
```

```python
import os
print(os.getcwd())  # Muestra el directorio de trabajo actual
```

Otros módulos útiles

datetime

Para manipulación de fechas y horas.

re

Para trabajar con expresiones regulares.

urllib

Para acceder a recursos en internet.

random

Para generación de números aleatorios.

json

Para manipular datos en formato JSON.

csv

Para leer y escribir archivos CSV.

Flexibilidad de la librería estándar

La librería estándar es suficientemente completa para manejar gran parte de las necesidades de un proyecto sin recurrir a librerías externas, lo cual agiliza el desarrollo y simplifica la distribución del software.

alkemy

Live Coding

¿En qué consistirá la Demo?

Vamos a construir una rutina para calcular el índice de masa corporal (IMC), separando la lógica en funciones y utilizando funciones de entrada/salida, cálculo y clasificación.

1. *Crear función calcular_imc(peso, altura)*
2. *Crear función clasificar_imc(imc) que devuelva la categoría*
3. *Utilizar input() para obtener datos del usuario*
4. *Usar float() para convertir texto en números*
5. *Mostrar resultado con print() y mensajes personalizados*
6. *Documentar cada función con docstrings*
7. *Separar código principal del resto usando if __name__ == "__main__"*
8. *Mostrar cómo importar el módulo desde otro script (demo)*

Tiempo: 25 Minutos

#Momentode Preguntas...

¿Cuál es la diferencia entre una función pre-construida y una personalizada?

¿Qué ventajas ofrece modularizar el código con funciones?

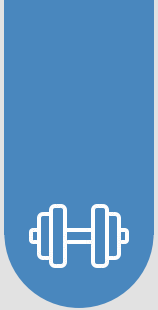
¿Cómo se estructura una función con parámetros y retorno?

¿Por qué es importante documentar las funciones con docstrings?

Momento:

Time-out!

⏰5 -10 min.



Ejercicio N° 1

Mi rutina saludable

Mi rutina saludable

Contexto: 🙌

Queremos ayudarte a automatizar el cálculo del Índice de Masa Corporal (IMC) y su clasificación según valores de referencia, usando funciones de Python.

Consigna: ✍️

Desarrollar un programa que cumpla con los siguientes requerimientos:

1. Crear una función personalizada llamada calcular_promedio que reciba como parámetros tres notas numéricas y devuelva el promedio de las mismas.
2. Crear una segunda función llamada mostrar_resultado que reciba el promedio y retorne un mensaje indicando si el estudiante aprueba (promedio ≥ 6) o no.
3. Desde el código principal:

- Pedir al usuario su nombre y las tres notas utilizando la función input()
- Convertir las entradas numéricas con float()
- Llamar a ambas funciones en secuencia: primero calcular_promedio(), luego mostrar_resultado()
- Mostrar el mensaje final al usuario utilizando print()

Tiempo 🕛**:** *40 Minutos*

Paso a paso: ⚙️

1. Solicitar peso (en kg) y altura (en metros) al usuario con input()
2. Crear función calcular_imc(peso, altura) → fórmula: peso / altura²
3. Crear función clasificar_imc(imc) con retornos tipo:

- <18.5 → Bajo peso
- 18.5–24.9 → Normal
- 25–29.9 → Sobrepeso
- 30+ → Obesidad

Mostrar resultados con print() incluyendo categoría y valor
Usar round() para redondear el resultado a 2 decimales
Separar funciones y lógica principal para fomentar modularidad

¿Alguna consulta?

alkemy

Resumen

¿Qué logramos en esta clase?

- ✓ ***Utilizamos funciones integradas como input(), print(), type() y round()***
- ✓ ***Creamos funciones propias con parámetros, retorno y documentación***
- ✓ ***Organizamos el código para que sea reutilizable y más claro***
- ✓ ***Usamos módulos para estructurar funcionalidades reutilizables***
- ✓ ***Aplicamos buenas prácticas como if __name__ == "__main__" y docstrings***

alkemy

¡Ponte a prueba!

Momento de ejercitación

Te invitamos a aprovechar esta última sección del espacio sincrónico para realizar de manera individual las ***actividades disponibles en la plataforma.*** *Estas propuestas son claves para afianzar lo trabajado y* ***forman parte obligatoria del recorrido de aprendizaje.***

💪 Análisis de caso ——— 💪 Selección Múltiple

💪 Comprensión lectora

Si al resolverlas surge alguna duda, compartela o tráela al próximo encuentro sincrónico.

alkemy

¡Muchas gracias!

alkemy