Recibe una cálida:

¡Bienvenida!

Te estábamos esperando 😁

alkemy

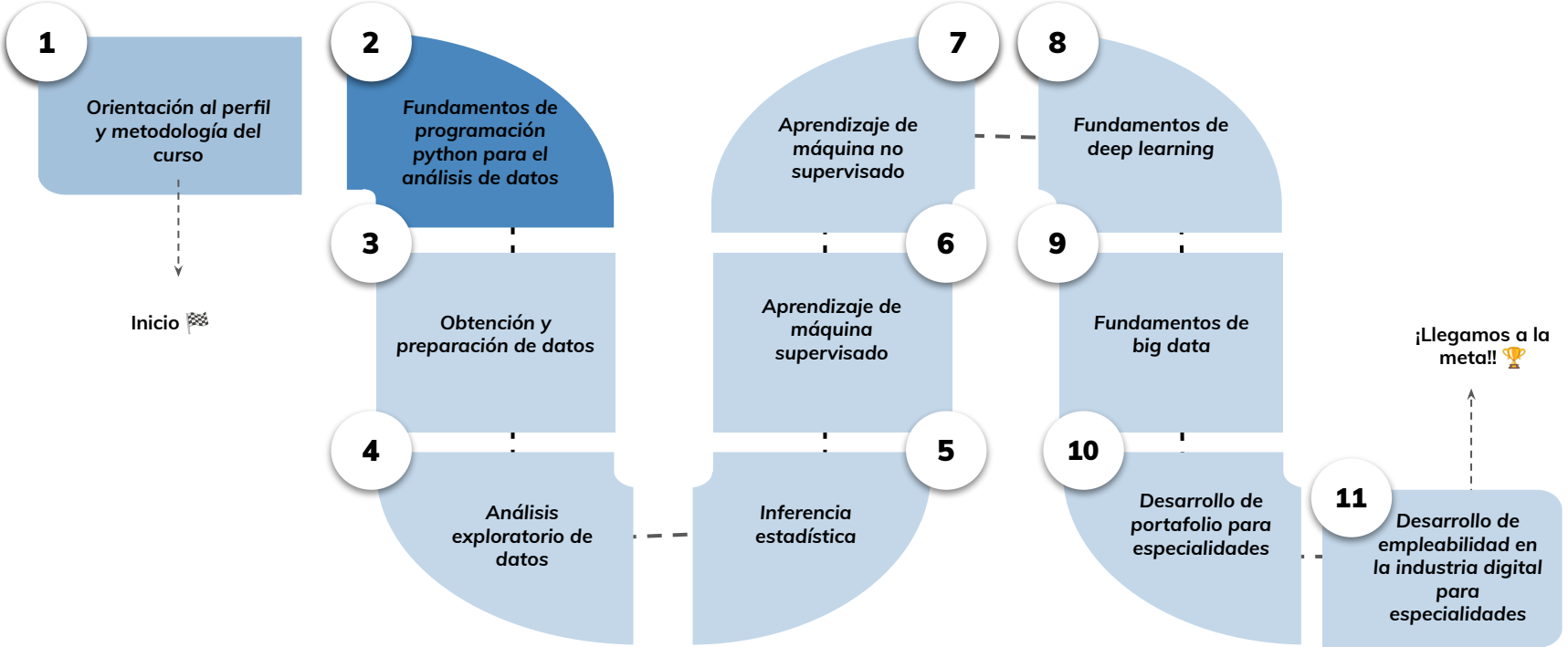
Clase grabada

Funciones y módulos- Parte 2

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

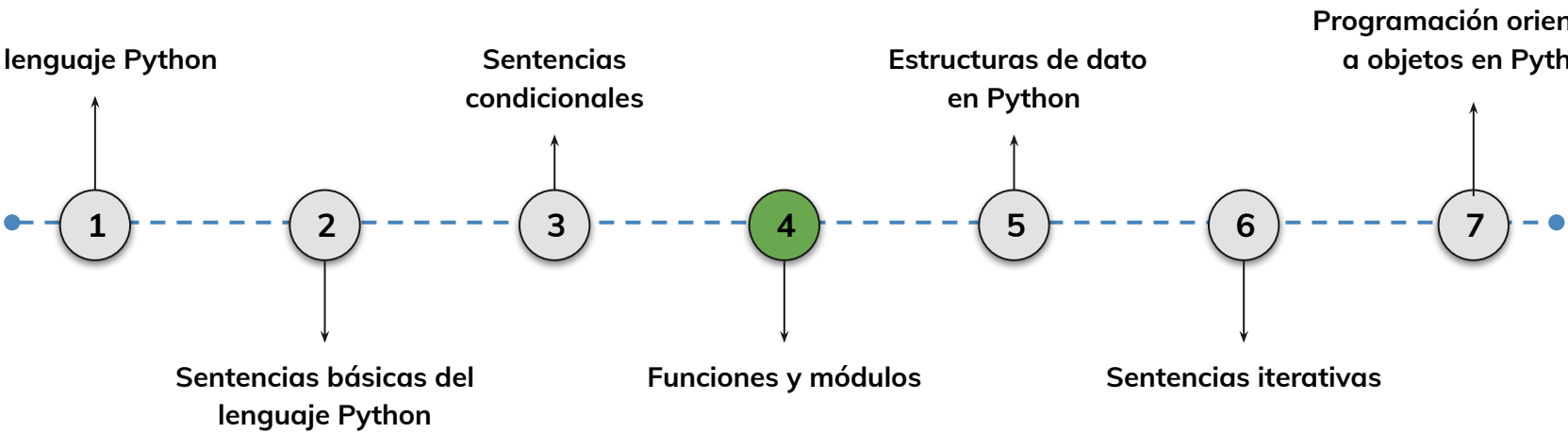
¿Cuáles skills conforman el programa? **Fundamentos de Ciencia de Datos**



alkemy

Roadmap de lecciones

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



alkemy

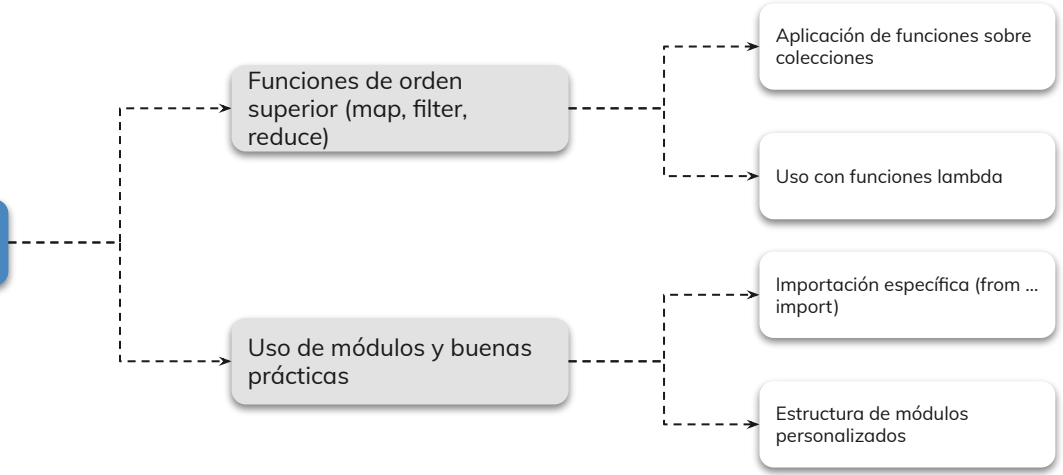
Learning Path

¿Cuáles temas trabajaremos hoy?

4.

Uso combinado de funciones preconstruidas, personalizadas y módulos para resolver problemas con Python.

Estaremos repasando conceptos previos sobre funciones y aplicamos nuevas herramientas del lenguaje Python como map(), filter() y reduce(). También repasaremos módulos estándar para resolver tareas más elaboradas y mejorar la reutilización del código.



alkemy

Objetivos de aprendizaje

¿Qué aprenderás?

- Aplicar funciones de orden superior (map, filter, reduce) para transformar y filtrar datos
- Utilizar funciones lambda para definir lógicas simples y temporales
- Importar funciones específicas desde módulos estándar (functools, math, etc.)
- Modularizar el código en funciones reutilizables y legibles
- Resolver problemas reales usando una estructura de programa limpia y mantenible

Repaso clase anterior

¿Quedó alguna duda?

En la clase anterior trabajamos 📚:

- → *Comprendimos la diferencia entre funciones preconstruidas y personalizadas*
- → *Definimos funciones con parámetros y retorno de valores*
- → *Usamos input() y print() junto con funciones propias para interactuar con el usuario*
- → *Resolvimos un problema educativo calculando promedios de forma modular*

Funciones y módulos

Importación de módulos en Python

Para usar un módulo en Python, se utiliza la palabra clave import, seguida del nombre del módulo. Esto permite acceder a las funciones, variables y clases definidas en dicho módulo.

```
# Importación básica
import math# Usar una función del módulo
resultado = math.sqrt(16)
print(resultado) # Imprime: 4.0
```

Importación con alias

Es posible usar un alias para simplificar el nombre del módulo.

Por ejemplo:

```
# Importar con alias
import numpy as np# Usar el alias para acceder a
funciones
array = np.array([1, 2, 3, 4, 5])print(array)
```

```
import math as m
print(m.sqrt(16))  # Salida: 4.0
```

alkemy

Importación de funciones específicas

Para importar solo una función o clase de un módulo, se utiliza from. Por ejemplo:

```
# Importar funciones específicas
from math import sqrt, pi# Usar las funciones
directamente
resultado = sqrt(16)print(resultado)  # Imprime:
4.0print(pi)  # Imprime: 3.141592653589793
```

```
from math import sqrt
print(sqrt(25))  # Salida: 5.0
```

Ventajas de importar solo lo necesario

Legibilidad

Mejora la legibilidad del código al usar directamente los nombres de las funciones sin el prefijo del módulo.

Eficiencia

Reduce la memoria utilizada al cargar solo las funciones necesarias en lugar de todo el módulo.

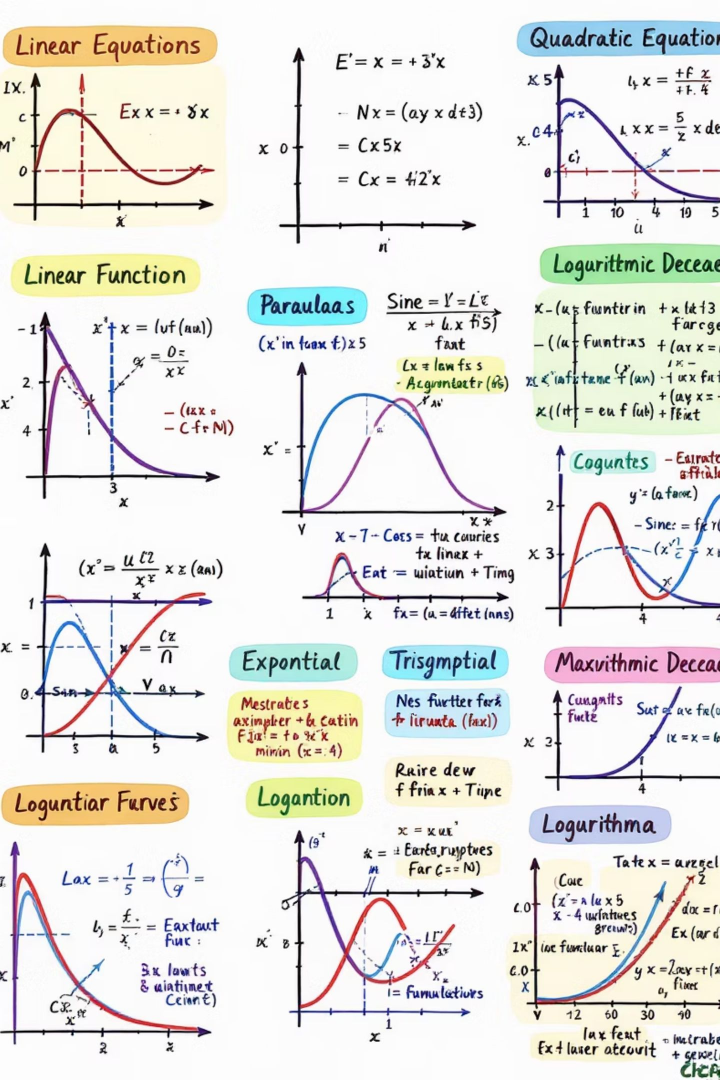
Claridad

Hace explícito qué funciones específicas se están utilizando del módulo.

Ejemplo completo de importación

```
# Diferentes formas de importar
import mathfrom datetime import datetime
import os as sistema
from random import randint, choice# Uso de las
importaciones
raiz = math.sqrt(25)ahora = datetime.now()archivos =
sistema.listdir('.')
numero = randint(1, 10)
fruta = choice(['manzana', 'pera', 'uva'])
print(raiz, ahora, archivos, numero, fruta)
```

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



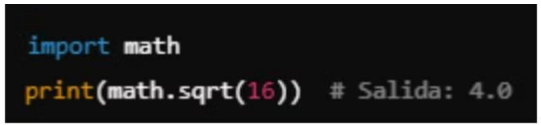
El módulo math

El módulo math proporciona funciones para operaciones matemáticas avanzadas, como potencias, raíces y logaritmos.

alkemy

Operaciones básicas con math

```
import math# Raíz cuadradaraiz = math.sqrt(16)
print(raiz)  # 4.0# Potencia
potencia = math.pow(2, 3)print(potencia)  # 8.0# Logaritmo
natural
logaritmo = math.log(10)print(logaritmo)  #
2.302585092994046# Logaritmo base 10
logaritmo10 = math.log10(100)print(logaritmo10)  # 2.0
```



Funciones trigonométricas

Seno

math.sin(x) - Calcula el seno de x (en radianes).

Coseno

math.cos(x) - Calcula el coseno de x (en radianes).

Tangente

math.tan(x) - Calcula la tangente de x (en radianes).

Conversión

math.radians(x) - Convierte grados a radianes.

math.degrees(x) - Convierte radianes a grados.

alkemy

Constantes de math

π

math.pi

Valor de pi (3.141592653589793)

e

math.e

Valor de e (2.718281828459045)

τ

math.tau

Valor de tau (6.283185307179586), equivalente a 2π

∞

math.inf

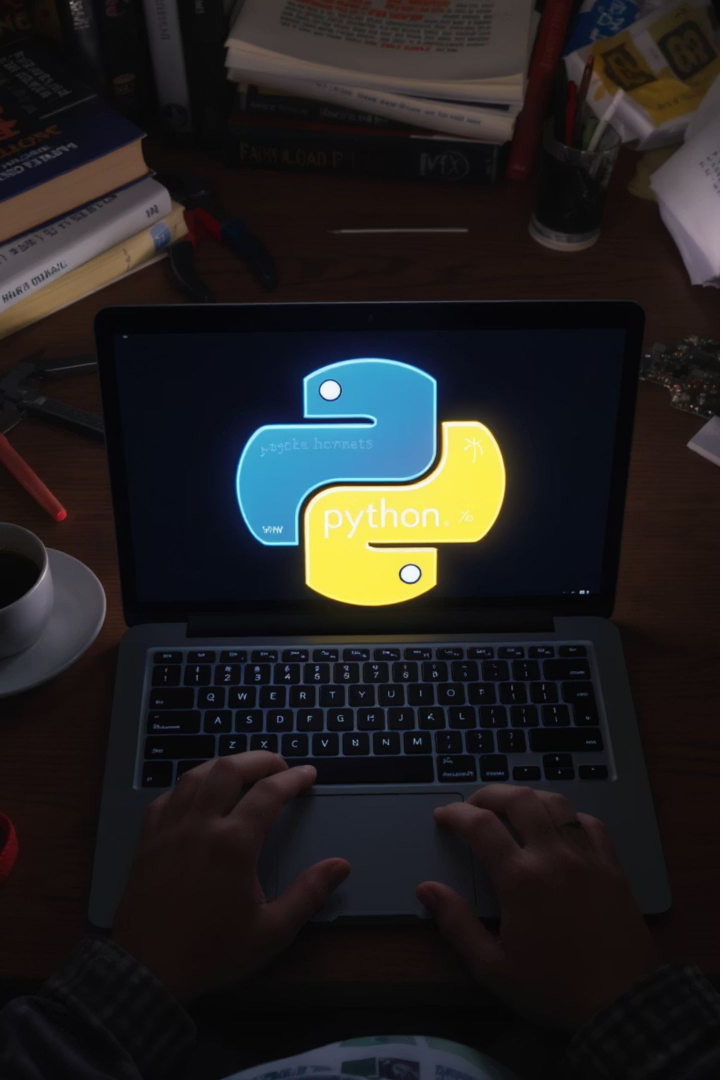
Valor positivo infinito

Operaciones de redondeo

math ofrece funciones de redondeo como ceil (redondeo hacia arriba) y floor (redondeo hacia abajo).

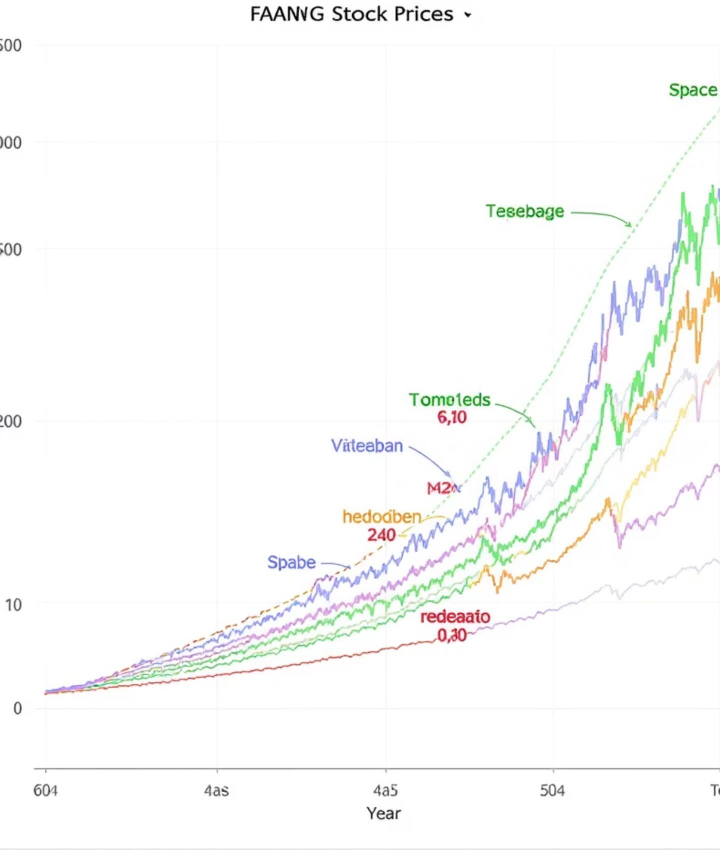
```
import math# Redondeo hacia arribatecho =
math.ceil(4.2)print(techo)  # 5# Redondeo hacia abajopiso
= math.floor(4.8)print(piso)  # 4# Truncamiento (elimina
la parte decimal)truncado = math.trunc(4.9)print(truncado)
# 4
```

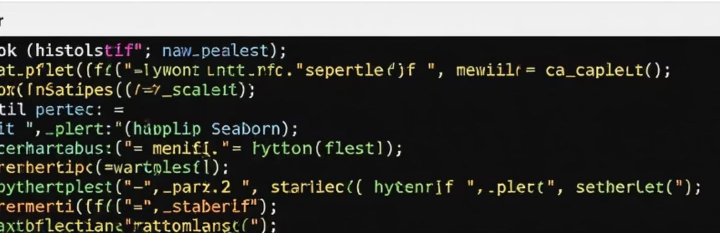
```
print(math.ceil(2.3))  # Salida: 3
print(math.floor(2.7))  # Salida: 2
```



Aplicación en cálculos científicos

Las funciones del módulo math son útiles en aplicaciones científicas y de ingeniería, donde se requiere precisión y eficiencia en cálculos complejos.





El módulo statistics

El módulo statistics ofrece funciones para cálculos estadísticos básicos, como media, mediana y moda.

alkemy

Funciones básicas de estadística

```
import statistics# Lista de datosdatos = [2, 5, 7, 9, 12, 5, 8, 9]# Media (promedio)media = statistics.mean(datos)print(f"Media: {media}")  # 7.125# Mediana (valor central)mediana = statistics.median(datos)print(f"Mediana: {mediana}")  # 7.5# Moda (valor más frecuente)moda = statistics.mode(datos)print(f"Moda: {moda}")  # 5
```

```python
import statistics as stats
datos = [10, 20, 20, 30, 40]
print(stats.mean(datos))  # Salida: 24.0
print(stats.median(datos))  # Salida: 20
print(stats.mode(datos))  # Salida: 20
```

Medidas de dispersión

Desviación estándar

statistics.stdev() - Calcula la desviación estándar de una muestra.

Varianza

statistics.variance() - Calcula la varianza de una muestra.

Desviación poblacional

statistics.pstdev() - Calcula la desviación estándar de una población.

Varianza poblacional

statistics.pvariance() - Calcula la varianza de una población.

Cálculos avanzados con statistics

```
import statisticsdatos = [2, 5, 7, 9, 12, 5, 8, 9]# Desviación estándardesviacion = statistics.stdev(datos)print(f"Desviación estándar: {desviacion}")# Varianzavarianza = statistics.variance(datos)print(f"Varianza: {varianza}")# Cuartilesq1 = statistics.quantiles(datos, n=4)[0] # Primer cuartilq2 = statistics.quantiles(datos, n=4)[1] # Segundo cuartil (mediana)q3 = statistics.quantiles(datos, n=4)[2] # Tercer cuartilprint(f"Cuartiles: {q1}, {q2}, {q3}")
```

Aplicación en ciencia de datos

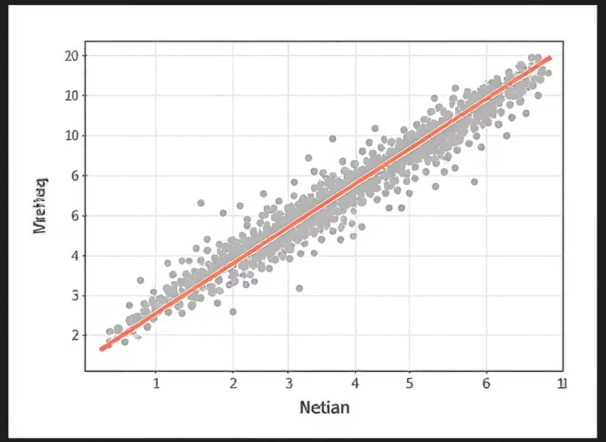
El módulo statistics es ideal para la ciencia de datos y la estadística, donde es necesario resumir y analizar grandes volúmenes de información.

alkemy

```
if meat maination megiet :
  fickil = V: StanMean_diegip:;

if tadf = meedial I
  riea:Cirkte·(iente_vl))
  risa:Cinkl/me(istartiple:(í());
  risa:(Stalltarl_Cocrisic; f);=)();
  risa:(Stalmart_Clerlation (= v())

  first mevision mules degilismeottet
  dlectotes {    {
      ff mear (dente_cleanets(c = 112); f|());
  }
  ticisl..IValrkiest_11)_();
  }
  santd_meorvations:
  }
  }
}
```



Facilidad de uso para principiantes

Con el módulo statistics, incluso los principiantes pueden calcular medidas estadísticas sin necesidad de herramientas complejas.

alkemy

Funciones de orden superior

Las funciones de orden superior son aquellas que aceptan otras funciones como argumentos o devuelven una función como resultado. En Python, esto permite programar de manera más flexible y reducir la repetición de código.

alkemy

Función map()

```
# Aplicar una función a cada elemento de una listanumeros = [1, 2, 3, 4, 5]# Usando map con una función definida
def cuadrado(x): return x ** 2
resultado = map(cuadrado, numeros)
print(list(resultado)) # [1, 4, 9, 16, 25]
# Usando map con una función lambda
resultado = map(lambda x: x * 3, numeros)
print(list(resultado)) # [3, 6, 9, 12, 15]
```

Función filter()

```
# Seleccionar elementos que cumplen una condición
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]# Filtrar números pares
pares = filter(lambda x: x % 2 == 0, numeros)
print(list(pares)) # [2, 4, 6, 8, 10]# Filtrar números mayores que 5
mayores = filter(lambda x: x > 5, numeros)print(list(mayores)) # [6, 7, 8, 9, 10]
```

Función reduce()

```
from functools import reduce# Combinar elementos de una lista en un solo valor
numeros = [1, 2, 3, 4, 5]# Sumar todos los elementos
suma = reduce(lambda x, y: x + y, numeros)
print(suma) # 15# Multiplicar todos los elementos
producto = reduce(lambda x, y: x * y, numeros)print(producto) # 120
```

Ventajas de las funciones de orden superior

Código conciso

Permiten realizar operaciones complejas con menos líneas de código.

Legibilidad

Mejoran la legibilidad al expresar claramente la intención del código.

Reutilización

Facilitan la reutilización de lógica en diferentes contextos.

Flexibilidad

Ofrecen mayor flexibilidad al permitir pasar comportamiento como argumento.

Expresiones lambda

Las expresiones lambda son funciones anónimas que se definen en una línea. Combinadas con map y filter, permiten escribir código conciso y eficiente.

```
# Sintaxis básica de lambda# lambda argumentos: expresión# Ejemplos de expresiones lambda
sumar = lambda x, y: x + y
print(sumar(5, 3)) # 8
multiplicar = lambda x, y, z: x * y * z
print(multiplicar(2, 3, 4)) # 24
# Lambda con condicionales_
par = lambda x: "Par" if x % 2 == 0 else "Impar"print(es_par(4)) # Par
print(es_par(7)) # Impar
```

Ejemplo completo de filter y reduce

```
from functools import reduce# Lista de productos con precio y stockproductos = [ {"nombre": "Laptop", "precio": 1200, "stock": 5}, {"nombre": "Teléfono", "precio": 800, "stock": 10}, {"nombre": "Tablet", "precio": 500, "stock": 0}, {"nombre": "Monitor", "precio": 300, "stock": 8}, {"nombre": "Teclado", "precio": 100, "stock": 15}]# Filtrar productos disponibles (stock > 0)disponibles = filter(lambda p: p["stock"] > 0, productos)# Calcular el valor total del inventariovalor_total = reduce(lambda acum, p: acum + (p["precio"] * p["stock"]), disponibles, 0)print(f"Valor total del inventario: ${valor_total}")
```

Combinación de funciones de orden superior

```
# Combinar map, filter y reducefrom functools import reducenumeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]# Filtrar números pares, elevarlos al cuadrado y sumarlosresultado = reduce( lambda x, y: x + y, map( lambda x: x ** 2, filter(lambda x: x % 2 == 0, numeros) ))print(resultado) # 2² + 4² + 6² + 8² + 10² = 4 + 16 + 36 + 64 + 100 = 220
```

Creación de módulos personalizados

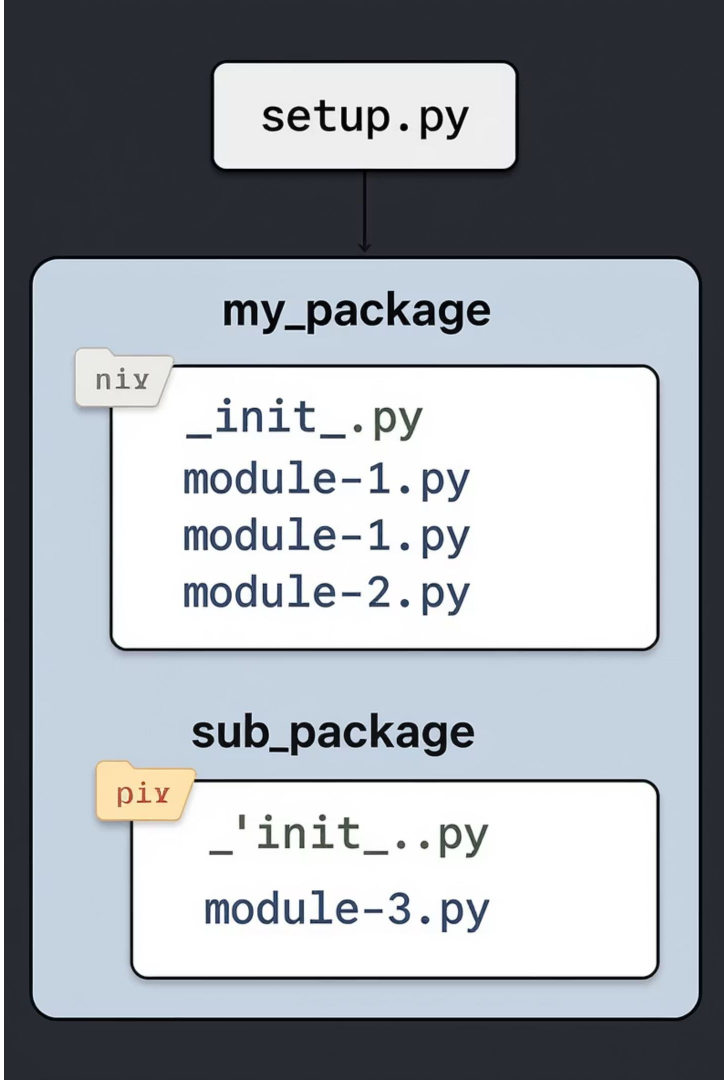
Crear módulos personalizados es sencillo en Python. Solo necesitas crear un archivo .py con tus funciones y luego importarlo en otros scripts.

```
# Archivo: mi_modulo.pydef saludar(nombre): return f"¡Hola, {nombre}!"def despedir(nombre): return f"¡Adiós, {nombre}!"PI = 3.14159GRAVEDAD = 9.8# En otro archivoimport mi_moduloprint(mi_modulo.saludar("Ana"))print(mi_modulo.PI)
```

Paquetes en Python

Un paquete es una carpeta que contiene varios módulos relacionados.

Para crear un paquete, se necesita un archivo especial llamado __init__.py (puede estar vacío) dentro de la carpeta.



Estructura de un paquete

```
mi_paquete/ __init__.py modulo1.py modulo2.py subpaquete/ __init__.py modulo3.py# Importación desde un paqueteimport mi_paquete.modulo1from mi_paquete import modulo2from mi_paquete.subpaquete import modulo3
```

Instalación de paquetes externos

Python permite instalar paquetes externos usando pip, el gestor de paquetes de Python. Estos paquetes extienden la funcionalidad de Python con herramientas especializadas.

```
# Instalar un paquetepip install numpy# Instalar una versión específicapip install pandas==1.3.0# Actualizar un paquetepip install --upgrade matplotlib# Listar paquetes instaladospip list
```

Paquetes populares en Python

NumPy

Para cálculos numéricos y operaciones con arrays.

Matplotlib

Para creación de gráficos y visualizaciones.

Flask

Para desarrollo web ligero y aplicaciones web.

Pandas

Para análisis y manipulación de datos estructurados.

Scikit-learn

Para aprendizaje automático y minería de datos.

SQLAlchemy

Para trabajar con bases de datos relacionales.

alkemy

Buenas prácticas con funciones y módulos

Nombres descriptivos

Usar nombres claros y descriptivos para funciones y módulos.

Documentación

Documentar todas las funciones con docstrings explicando su propósito, parámetros y valores de retorno.

Principio de responsabilidad única

Cada función debe hacer una sola cosa y hacerla bien.

Organización lógica

Agrupar funciones relacionadas en el mismo módulo.

Importaciones específicas

Importar solo lo que se necesita para evitar conflictos y mejorar el rendimiento.

Conclusión

Las funciones y módulos en Python son esenciales para desarrollar código modular, reutilizable y organizado. Con funciones, los programadores pueden encapsular lógica específica en bloques reutilizables, mientras que los módulos permiten estructurar el código en componentes bien definidos.

La librería estándar de Python y sus módulos preconstruidos, como math y statistics, ofrecen herramientas potentes y listas para usar en una variedad de tareas, desde operaciones matemáticas hasta cálculos estadísticos.

Las funciones de orden superior permiten escribir código más conciso y eficiente, promoviendo un estilo de programación funcional que simplifica el manejo de datos y mejora la legibilidad.

El dominio de funciones y módulos es un paso esencial para avanzar hacia la programación avanzada en Python, permitiendo que los programadores desarrollen proyectos más complejos y especializados.

Live Coding

¿En qué consistirá la Demo?

Vamos a construir un sistema simple de inventario para una tienda. Utilizaremos funciones personalizadas y funciones de orden superior como map(), filter() y reduce() para analizar el stock y calcular el valor total.

1. *Definir una lista de diccionarios representando productos (nombre, precio, stock)*
2. *Aplicar filter() con lambda para obtener productos disponibles*
3. *Usar map() para transformar nombres a mayúsculas o ajustar precios*
4. *Aplicar reduce() (de functools) para calcular el valor total del inventario*
5. *Mostrar los resultados usando print()*
6. *Crear funciones personalizadas para modularizar cada parte*
7. *Usar from functools import reduce como ejemplo de importación específica*
8. *Demostrar cómo reusar esas funciones en otros contextos o scripts*

Tiempo: 25 Minutos

#Momentode Preguntas...

¿?

¿Qué diferencia hay entre map() y filter()?

¿?

¿Para qué sirve la función lambda en una operación?

¿?

¿Cómo se importa solo una función desde un módulo en Python?

¿?

¿Qué ventajas tiene dividir un programa en funciones pequeñas?

Momento:

Time-out!

⏰5 -10 min.

Ejercicio N° 1

Inventario inteligente con funciones y módulos

Inventario inteligente con funciones y módulos

Contexto: 🙌

Un sistema de stock necesita analizar una lista de productos disponibles. Cada producto tiene nombre, precio y cantidad. Se busca filtrar los productos que tienen stock positivo, calcular su valor total (precio × cantidad) y mostrar la información de forma ordenada, reutilizando código mediante funciones.

Consigna: ✍️

Crear una lista de productos (nombre, precio y stock). Usar:

- filter() para quedarte solo con los productos con stock disponible.
- reduce() (de functools) para calcular el valor total del stock disponible.
- Funciones personalizadas para organizar el código.
- print() para mostrar los resultados finales.
- Todo debe estar modularizado con funciones bien definidas.

Tiempo 🕛**:** *40 Minutos*

Inventario inteligente con funciones y módulos

Paso a paso: ⚙️

1. Crear una lista de productos, donde cada producto es un diccionario con:
   - "nombre" (str), "precio" (float), "stock" (int)

2. Crear las siguientes funciones:
   - filtrar_disponibles(lista) → devuelve productos con stock > 0 usando filter()
   - calcular_valor_total(lista) → usa reduce() para obtener suma de precio × stock
   - mostrar_productos_mayus(lista) → usa map() para imprimir nombres

3. En el cuerpo principal:

- Llamar a las funciones y mostrar resultados con print()
- Redondear el valor total a 2 decimales con round()
- Modularizar correctamente todo el código
- Mostrar cómo importar solo reduce desde functools

¿Alguna consulta?

alkemy

Resumen

¿Qué logramos en esta clase?

- ✓ ***Aplicamos funciones de orden superior (map, filter, reduce) en problemas concretos***
- ✓ ***Usamos funciones lambda para lógicas simples***
- ✓ ***Importamos funciones específicas desde módulos estándar***
- ✓ ***Reforzamos el uso de funciones personalizadas bien estructuradas***
- ✓ ***Creamos una solución modular para un caso real usando Python***

¡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