PYTHON1: DESARROLLO BACKEND CON PYTHON, POO Y LA ERA DE LAS APLICACIONES WEB

3. PROGRAMACIÓN ORIENTADA A OBJETOS

FEDISAL | FORMÁS
CENTRO DE FORMACIÓN AVANZADA PARA PROFESIONALES

INCAF
Instituto Nacional de Capacitación y Formación

CONTENIDOS

1. Funciones.
2. Clases y objetos.
3. Encapsulamiento.
4. Herencia.
5. Polimorfismo

PROGRAMACIÓN ORIENTADA A OBJETOS

- Este modo o paradigma de programación nos permite organizar el código de una manera que se asemeja bastante a como pensamos en la vida real
- Utilizando las famosas clases.
- Estas nos permiten agrupar un conjunto de variables y funciones que veremos a continuación

PASO POR VALOR/REFERENCIA?

Si conoces lenguajes de programación como C, los conceptos de paso por valor o referencia te resultarán familiares:

Los tipos **inmutables** son pasados por **valor**, por lo tanto dentro de la función se accede a una copia y no al valor original.

Los tipos **mutables** son pasados por **referencia**, como es el caso de las listas y los diccionarios. Algo similar a como C pasa las array como punteros.

FUNCIONES

- Anteriormente hemos usado funciones nativas que vienen con Python como len() para calcular la longitud de una lista, pero al igual que en otros lenguajes de programación, también podemos definir nuestras propias funciones. Para ello hacemos uso de def.
- Cualquier función tendrá un **nombre**, unos **argumentos de entrada**, un **código** a ejecutar y unos **parámetros de salida**
- **Reusabilidad**
- **Modularidad**

FUNCIONES

- podemos definir nuestras propias funciones. Para ello hacemos uso de def.

```
def nombre_funcion(argumentos):
    código
    return retorno
```

- Cualquier función tendrá un nombre, unos argumentos de entrada, un código a ejecutar y unos parámetros de salida.

OTROS TEMAS DE FUNCIONES

- Valores por default
- Argumentos variables
- Recursividad

MODULOS

- librerías de funciones y clases para ser importadas y utilizadas en otro programa.
- Un módulo es un archivo que contiene definiciones y declaraciones de Python.
- El nombre del archivo es el nombre del módulo con el sufijo .py añadido.
- Dentro de un módulo, el nombre del módulo (como una cadena) está disponible como el valor de la variable global __name__.
- Por ejemplo, un archivo llamado aritme.py se importaría así:

```
import aritme
```

MODULOS

- Los módulos pueden importar otros módulos.
- Es habitual, pero no obligatorio, colocar todas las declaraciones de importación al comienzo de un módulo (o script, para el caso).
- Los nombres de los módulos importados se colocan en la tabla de símbolos global del módulo de importación.
- Existe una variante de la declaración de importación que importa nombres de un módulo directamente a la tabla de símbolos del módulo de importación. Por ejemplo:

```
from aritme import suma, resta

from aritme import *

import aritme as ari

from aritme import suma as sum
```

PACKAGES

- Los paquetes son una forma de estructurar el espacio de nombres de los módulos de Python mediante el uso de "nombres de módulos con puntos".
- Por ejemplo, el nombre del módulo A.B designa un submódulo llamado B en un paquete llamado A. Al igual que el uso de módulos evita que los autores de diferentes módulos tengan que preocuparse por los nombres de las variables globales de los demás, el uso de nombres de módulos con puntos salva a los autores de paquetes de varios módulos como NumPy o Pillow de tener que preocuparse por los nombres de los módulos de cada uno.

PACKAGES

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

- import sound.effects.echo
- import sound.effects.surround
- from sound.effects import *
- from sound.effects.echo import echofilter

PACKAGES

- Cuando el intérprete ejecuta la declaración de importación anterior, busca mod.py en una lista de directorios ensamblados a partir de las siguientes fuentes:
- El directorio desde el que se ejecutó el script de entrada o el directorio actual si el intérprete se ejecuta de forma interactiva
- La lista de directorios contenidos en la variable de entorno PYTHONPATH, si está configurada. (El formato de PYTHONPATH depende del sistema operativo, pero debe imitar la variable de entorno PATH).
- Una lista dependiente de la instalación de directorios configurados en el momento de instalar Python

```
import sys

sys.path.append(r'D:\Desktop\Modulo1Python')

print(str(sys.path))
```

CLASES Y OBJETOS

- Clase: La descripción de un objeto (estados y comportamientos)
- Objeto: La instancia de una clase
- Instancia? La lectura de la clase para convertirla en un objeto
- Miembros de una clase: atributos y métodos
- Características de la POO?
- 1. Herencia (Clase base y derivada)
  - Herencia múltiple (Varios padres para una clase)
  - Clase Abstracta (no se puede instanciar)
- 2. Encapsulamiento
- 3. Polimorfismo (sobreescritura)

CONSTRUCTOR?

- Un método opcional con un nombre especial (__init__)
- Inicializa un objeto y sus variables de instancia (reserva la memoria)
- Método que se ejecuta en la creación del objeto
- Por ejemplo: Persona("juan")

ENCAPSULAMIENTO

- Se denomina encapsulamiento al ocultamiento del estado, es decir, de los datos miembro de un objeto de manera que solo se pueda cambiar mediante las operaciones definidas para ese objeto.
- De esta forma el usuario de la clase puede obviar la implementación de los métodos y propiedades para concentrarse solo en cómo usarlos. Por otro lado se evita que el usuario pueda cambiar su estado de maneras imprevistas e incontroladas.

ENCAPSULAMIENTO

- Dicho de otra manera, encapsular consiste en hacer que los atributos o métodos internos a una clase no se puedan acceder ni modificar desde fuera, sino que tan solo el propio objeto pueda acceder a ellos.
- Niveles de aislamiento típicos
  - Publico: accesible desde cualquier parte del programa
  - Protegido: accesible por la misma clase y clases en relación de herencia
  - Privado: accesible únicamente desde la misma clase

  - Elaborar ejemplos según indicaciones del instructor

HERENCIA

- La herencia es un proceso mediante el cual se puede crear una clase hija que hereda de una clase padre.
- Compartiendo sus métodos y atributos.
- Además de ello, una clase hija puede sobrescribir los métodos o atributos, o incluso definir unos nuevos.
- El principio DRY (Don't Repeat Yourself) consiste en no repetir código de manera innecesaria.
- la función super() nos permite acceder a los métodos de la clase padre desde una de sus hijas
- Todas las clases heredan de la clase Object
- Elaborar ejemplos según indicaciones del instructor

POLIMORFISMO

- Polimorfismo: nombres iguales comportamientos diferentes.
- El concepto de *polimorfismo* (del griego *muchas formas*) implica que si en una porción de código se invoca un determinado método de un objeto, podrán obtenerse distintos resultados según la clase del objeto. Esto se debe a que distintos objetos pueden tener un método con un mismo nombre, pero que realice distintas operaciones
- Es prácticamente sobrescribir el comportamiento de un método en la clase derivada en una relación de herencia.

POR EJEMPLO

```
class Padre(object):
    def __init__(self):
        self.value = 4
    def get_value(self):
        return self.value
class hijo(Parent):
    def get_value(self):
        return self.value + 1
```

Elaborar ejemplos según indicaciones del instructor

PREGUNTAS?

- Gracias por su atención!!!

FEDISAL | FORMÁS CENTRO DE FORMACIÓN AVANZADA PARA PROFESIONALES

INCAF Instituto Nacional de Capacitación y Formación


thank you
merci
gracias
danke
謝謝
ngiyabonga
teşekkür ederim
tapadh leat
go raibh maith agat
obrigado
dziękuję
bedankt
спасибо
mauruuru
hvala
sukriya
kop khun krap
terima kasih
dakujem
arigatô
grazie
takk
dank je
mochchakkeram
merci
рахмат
Баярлалаа
faafetai lava
kiitos
dankie
dhanyavad
nanni
nandri
bayarlalaa
gràçie
enkosi
köszönöm
vinaka
спасибі
blagodaram
mersi
kia ora
barka
welalin
tack
spas
misaotra
matondo
paldies
grazzi
mahalo
djiere dieuf
tau
akun
dankon
ačiū
chnorakaloutioun
gratias ago
gràcies
sulpáy
taiku
sagolun
sobodi
děkuji
mèsi
didi madloba
najis tuke
ありがとう
tanemirt
rahmet
rahmat
kam sah hamnida
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ
diolch
dhanyavadagalu
shukriya
mercé
мерси
trugarez
дякую
mamnun
chokrane
murakoze
хвала
asante
manana
obrigada
tenki