CURSOS polotic misiones

PROGRAMACIÓN WEB

CON PYTHON Y JAVASCRIPT

CLASE 4

CLASE 4
INTRODUCCIÓN AL LENGUAJE PYTHON

CURSOS polotic MISIONES

python™

CLASE 4
HOY

CURSOS polotic misiones

- Programación Orientada a Objetos con Python
- Programación Funcional
- Tratamiento de Excepciones

CLASE 4

CURSOS polotic MISIONES

Programación Orientada a Objetos

CLASE 4
PROGRAMACION ORIENTADA A OBJETOS

CURSOS polotic misiones

- La programación orientada a objetos es un **paradigma de programación**, o una forma de pensar sobre la programación, que **se centra en objetos** que pueden almacenar información y realizar acciones.

CLASE 4
PROGRAMACION ORIENTADA A OBJETOS

CURSOS polotic misiones

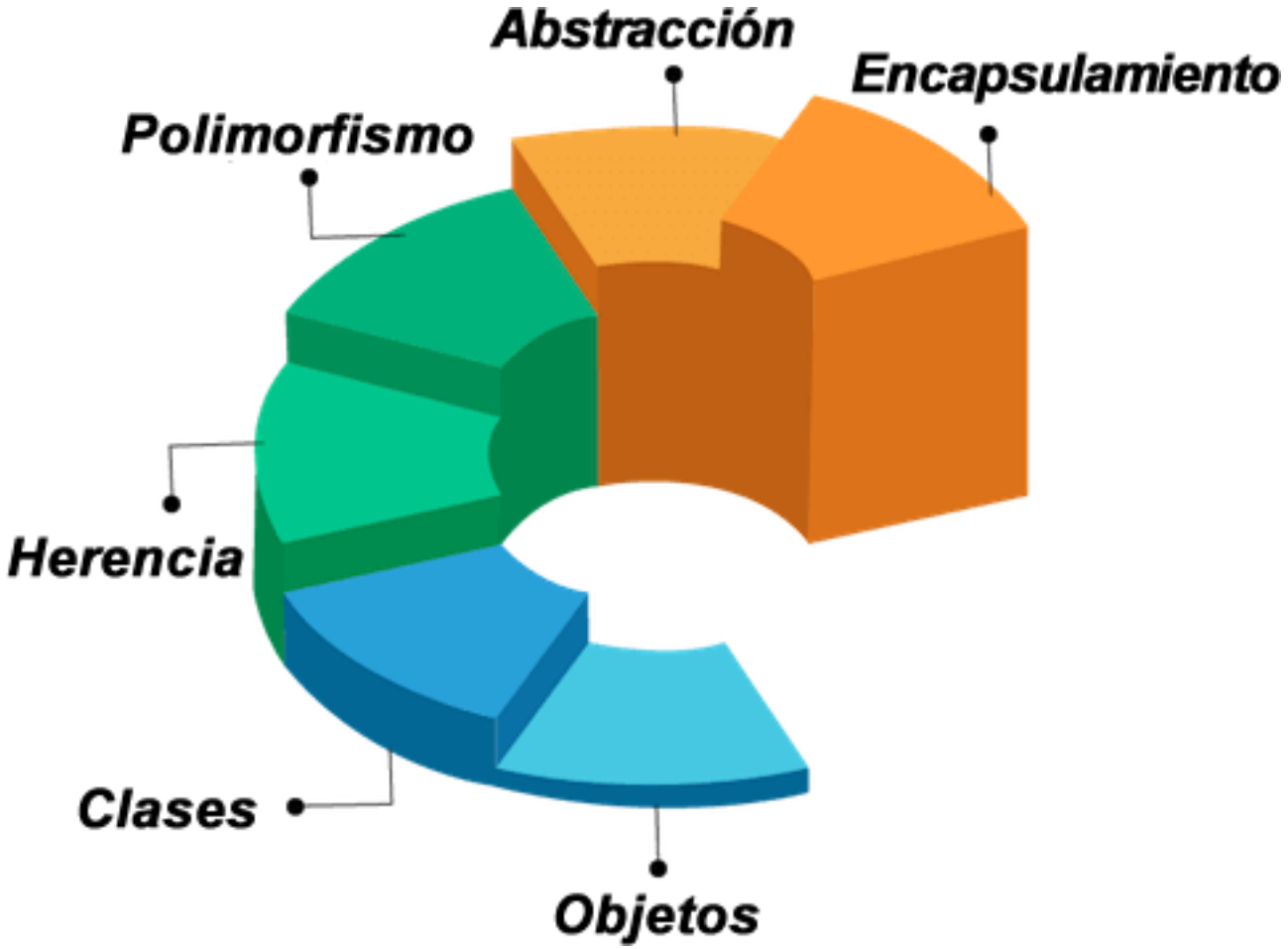
CLASE:
Vehículo

OBJETO:
Automóvil

Atributos/Propiedades:
marca
modelo
color

Métodos:
inyectar_combustible
frenar
cambiar_marcha

CLASE 4
PROGRAMACION ORIENTADA A OBJETOS



CLASE 4
PROGRAMACION ORIENTADA A OBJETOS

CURSOS polotic misiones

- **Enlaces de interés**:
  - Wikipedia: https://es.wikipedia.org/wiki/Programación_orientada_a_objetos
  - MIT: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-01sc-introduction-to-electrical-engineering-and-computer-science-i-spring-2011/unit-1-software-engineering/object-oriented-programming/
  - Brown University: http://cs.brown.edu/courses/cs015/
  - Princeton: https://introcs.cs.princeton.edu/java/30oop/

CLASE 4
PROGRAMACION ORIENTADA A OBJETOS

CURSOS polotic misiones

- **Clases**: ya hemos visto algunos tipos diferentes de variables en Python, pero *¿qué pasa si queremos crear nuestro propio tipo*?
- Una clase de Python es esencialmente **una plantilla para un nuevo tipo de objeto** que puede almacenar información y realizar acciones. Aquí hay una clase que define un punto bidimensional:

```
class Punto():
        # Un método que define como crear un punto:
        def __init__(self, x, y):
                self.x = x
                self.y = y
```

CLASE 4
PROGRAMACION ORIENTADA A OBJETOS

CURSOS polotic misiones

```
class Punto():
        # Un método que define como crear un punto:
        def __init__(self, x, y):
                self.x = x
                self.y = y
```

- Ten en cuenta que en el código anterior, usamos la palabra clave `self` para representar el objeto con el que estamos trabajando actualmente.
- `self` debería ser el primer argumento para cualquier método dentro de una clase Python.

Clases vs Instancias

- Las clases se utilizan para crear estructuras de datos definidas por el usuario. Las clases definen funciones llamadas métodos, que identifican los comportamientos y acciones que un objeto creado a partir de la clase puede realizar con sus datos.
- Una clase es un modelo de cómo se debe definir algo. En realidad, no contiene ningún dato. La clase Perro especifica que un nombre y una edad son necesarios para definir un perro, pero no contiene el nombre o la edad de ningún perro específico.
- Dicho de otra manera, una clase es como un formulario o cuestionario. Una instancia es como un formulario que se ha llenado con información. Al igual que muchas personas pueden completar el mismo formulario con su propia información única, se pueden crear muchas instancias a partir de una sola clase.

```
class Punto():
        # Un método que define como crear un punto:
        def __init__(self, x, y):
                self.x = x
                self.y = y
```

- Ahora, veamos cómo podemos usar la clase de arriba para crear un objeto:

INSTANCIA

```
p = Punto(2, 8)
print(p.x)
print(p.y)
```

CLASE 4
PROGRAMACION ORIENTADA A OBJETOS

CURSOS polotic misiones

- Ahora, veamos un ejemplo más interesante en el que en lugar de almacenar solo las coordenadas de un punto, creamos una clase que representa el vuelo de una aerolínea:

```
class Vuelo():
    # Metodo para crear un nuevo vuelo con una capacidad dada
    def __init__(self, capacidad):
        self.capacidad = capacidad
        self.pasajeros = []

    # Metodo para agregar un pasajero al vuelo:
    def agregar_pasajero(self, nombre):
        self. pasajeros.append(nombre)
```

- Sin embargo, esta clase tiene fallas porque aunque establezcamos una capacidad, aún podríamos agregar demasiados pasajeros. Modifiquemos para que antes de agregar un pasajero, verifiquemos si hay espacio en el vuelo:

```
class Vuelo():
    # Metodo para crear un nuevo vuelo con una capacidad dada
    def __init__(self, capacidad):
        self.capacidad = capacidad
        self.pasajeros = []

    # Metodo para agregar un pasajero al vuelo:
    def agregar_pasajero(self, nombre):
        if not self.asientos_disponibles():
            return False
        self.pasajeros.append(nombre)
        return True
    # Metodo para retornar el numero de asientos disponibles
    def asientos_disponibles(self):
        return self.capacidad - len(self.pasajeros)
```

Esto funciona porque en Python, el número 0 se puede interpretar como `False`, y también podemos usar la palabra clave `not` para significar lo opuesto a la siguiente declaración, por lo que `not True` es `False` y `not False` es `True`. Por lo tanto, si `asientos_disponibles` devuelve 0, toda la expresión se evaluará como `True`

CLASE 4
PROGRAMACION ORIENTADA A OBJETOS

CURSOS polotic misiones

- INSTANCIANDO LOS OBJETOS

Ahora, probemos la clase que hemos creado creando instancias de algunos objetos:

```
# Crear un nuevo vuelo con hasta 3 personas
unVuelo = Vuelo(3)
# Crear una lista de personas
personas = ["Harry", "Ron", "Hermione", "Ginny"]
# Intentar agregar cada persona de la lista al vuelo
for unaPersona in personas:
        if unVuelo.agregar_pasajero(unaPersona):
                print(f"Agregado {unaPersona} al vuelo satisfactoriamente")
        else: print(f"No hay asientos disponibles para  {unaPersona}")
```

CLASE 4

CURSOS polotic misiones

Programación Funcional

CLASE 4
PROGRAMACIÓN FUNCIONAL

CURSOS polotic misiones

- Además de admitir la Programación Orientada a Objetos, Python también admite el paradigma de **programación funcional**, en el que las funciones se tratan como valores como cualquier otra variable.
- En Programación Funcional las funciones son tratadas como ciudadanos de primera (first-class citizens), lo que significa que se le pueden asociar nombres, pueden ser pasadas como argumentos o retornadas desde otras funciones, como cualquier otro tipo de datos.
- Lo que permite que los programas sean escritos en estilos declarativos y compuestos, donde las pequeñas funciones son combinadas en manera modular.
- Surgió del ámbito científico universitario gracias al calculo lambda,

que era un sistema formal de computo basado en solamente funciones.

CLASE 4
PROGRAMACION FUNCIONAL

CURSOS polotic misiones

- **DECORATORS**

Una cosa que la programación funcional hace posible es la idea de un decorador, que es una función de orden superior que puede modificar otra función.

Por ejemplo, escriba un decorador que anuncie cuándo una función está a punto de comenzar y cuándo termina. Luego podemos aplicar este decorador usando un símbolo @.

```
def mi_mensaje(f):
        def wrapper():
                print("A punto de correr la función")
                f()
                print("Finalizada la función")
        return wrapper

@mi_mensaje
def hola():
        print("¡Hola, mundo!")

hola()
```

```
""" Salida:
A punto de correr la función
¡Hola, mundo!
Finalizada la función
"""
```

CLASE 4
PROGRAMACION FUNCIONAL

CURSOS polotic misiones

- **FUNCIONES LAMBDA**

Las funciones Lambda proporcionan otra forma de crear funciones en Python. Por ejemplo, si queremos definir la misma función cuadrado que hicimos en la clase anterior, podemos escribir:

```
cuadrado = lambda x: x * x
```

Donde la entrada está a la izquierda de `:` y la salida está a la derecha.

Esto puede ser útil cuando no queremos escribir una función completamente separada para un solo uso pequeño. Por ejemplo, si queremos ordenar algunos objetos en los que al principio no está claro cómo ordenarlos. Imagina que tenemos una lista de personas, pero con nombres y casas en lugar de solo nombres que deseamos ordenar:

```
personas = [
        {"nombre": "Harry", "casa": "Gryffindor"},
        {"nombre": "Cho", "casa": "Ravenclaw"},
        {"nombre": "Draco", "casa": "Slytherin"} ]

personas.sort()

print(personas)
```

- **FUNCIONES LAMBDA**

El codigo de nuestro diccionario produce un error:

```
Traceback (most recent call last):
  File "p:/Cursos Polo/Python con Javascript/Codigo Fuente/progfunc.py", line 5, in <module>
    personas.sort()
TypeError: '<' not supported between instances of 'dict' and 'dict'
```

Esto ocurre porque Python no sabe cómo comparar dos diccionarios para comprobar si uno es menor que el otro.

Podemos resolver este problema incluyendo un argumento `key` para la función `sort`, que especifica qué parte del diccionario deseamos usar para ordenar.

¡Veamos!

PROGRAMACIÓN WEB

CON PYTHON Y JAVASCRIPT

CURSOS polotic MISIONES

GRACIAS

programacionpolotic@gmail.com