SISTEMAS DE GESTIÓN EMPRESARIAL

Curso: 2º Desarrollo de Aplicaciones Multiplataforma
Centro: Atlantida CIDEP
Profesor: Daniel López Lozano

TEMA 4.
Introducción al Lenguaje Python

INDICE DE CONTENIDOS

DICCIONARIOS CONDICIONALES Y FUNCIONES

INTRODUCCION

- Los diccionarios sirven para no abusar tanto de los condicionales:

```
DISFRAZ_DEFECTO="PECERA"
adversario=input("Quien tienes enfrente")

if adversario.lower()=="loki":
    misterio="Lady Loki"
elif adversario.lower()=="hulk":
    misterio="Thanos"
elif adversario.lower()=="thor":
    misterio="Odin"
elif adversario.lower()=="superman":
    misterio="Darkseid"
else:
    misterio=DISFRAZ_DEFECTO

print(f"Misterio se disfraza:{misterio}")
```

```
disfraces_misterio={
        "superman":"Darkseid",
        "thor":"Odin",
        "loki":"Lady Loki",
        "hulk":"Thanos"
    }

adversario=input("Quien tienes enfrente")
misterio=disfraces_misterio.get(adversario.lower(),DISFRAZ_DEFECTO)
print(f"Misterio se disfraza:{misterio}")
```

LAS FUNCIONES SON TIPOS DE DATOS

- Podemos ampliar las posibilidades de los anteriores usando funciones en un diccionario

```
from random import choice
numeros=[1,2,3,4,5]
operacion=input("Que operacion quieres

if operacion == "sum":
    res=0
    for num in numeros:
        res+=num
elif operacion == "multi":
    res=1
    for num in numeros:
        res*=num
elif operacion == "ale":
    res=choice(numeros)
else:
    res="No existe"
```

LAS FUNCIONES SON TIPOS DE DATOS

- Permitiendo hacer acciones todo lo complejas que queramos:

```
def sumador(lista):
    res=0
    for num in lista:
        res+=num
    return res

def multiplicador(lista):
    res=0
    for num in lista:
        res*=num
    return res

def  aleatorio(lista):
    return choice(lista)

operaciones={}
operaciones["sum"]=sumador
operaciones["multi"]=multiplicador
operaciones["ale"]=aleatorio
```

LAS FUNCIONES SON TIPOS DE DATOS

- Para ejecutar lo anterior si error debemos usar el metodo get para mapas y una funcion lambda para el caso en que no se llame a un valor correcto:

```
nombre=input("Que operacion quieres realizar?")
operacion=operaciones.get(nombre,lambda lista: "No existe dicha operacion")
print(operacion(numeros))
```

ACCESO A FICHEROS

INTRODUCCION

- Hasta ahora hemos visto como interactuar con un programa a través del teclado (entrada de datos) y la terminal (salida),pero en la mayor parte de las aplicaciones reales tendremos que leer y escribir datos en ficheros.
- Al utilizar ficheros para guardar los datos estos perdurarán tras la ejecución del programa, pudiendo ser consultados o utilizados más tarde.

INTRODUCCION

Las operaciones más habituales conficheros son:

- Crear un fichero.
- Escribir datos en un fichero.
- Leer datos de un fichero.
- Borrar un fichero.

CREAR UN FICHERO

- Para crear un fichero nuevo se utilizala instrucción

`open(ruta, 'w')` : Crea el fichero con la ruta ruta, lo abre en modo escritura (el argumento ‘w’ significa write) y devuelve un objeto que lo referencia.

- Si el fichero indicado por la ruta ya existe en el sistema, se reemplazará por el nuevo.
- Una vez creado el fichero,para escribir datos en él se utiliza el método

`fichero.write(c)`:Escribe la cadena “c” en el fichero referenciado por fichero.

CERRAR UN FICHERO

- Para cerrar un fichero se utiliza el método:

`fichero.close()`: Cierra el fichero referenciado por el objeto fichero.

- Cuando se termina de trabajar con un fichero conviene cerrarlo, sobre todo si se abre en modo escritura, ya que mientras está abierto en este modo no se puede abrir por otra aplicación.
- Si no se cierra explícitamente un fichero, Python intentará cerrarlo cuando estime que ya no se va a usar más.

AÑADIR DATOS A UN FICHERO

- Si en lugar de crear un fichero nuevo queremos añadir datos a un fichero existente se debe utilizar la instrucción

`open(ruta, 'a')` : Abre el fichero con la ruta ruta en modo añadir (el argumento ‘a’ significa append) y devuelve un objeto que lo referencia.

- Una vez abierto el fichero, se utiliza el método de escritura anterior y los datos se añaden al final del fichero.

EJEMPLOS

```
f = open('bienvenida.txt', 'w')
f.write('¡Bienvenido a Python!')
f.close()

f = open('bienvenida.txt', 'a')
f.write('\n¡Hasta pronto!')
f.close()
```

LECTURA DE DATOS

- Para abrir un fichero en modo lectura se utiliza la instrucción:

`open(ruta, 'r')` : Abre el fichero con la ruta ruta en modo lectura (el argumento ‘r’ significa read) y devuelve un objeto que lo referencia.

- Una vez abierto el fichero,se puede leer todo el contenido del fichero o se puede leer línea a línea.

LECTURA DE DATOS

- `fichero.read()` :Devuelve todos los datos contenidos en fichero como una cadena de caracteres.
- `fichero.readlines()` : Devuelve una lista de cadenas de caracteres donde cada cadena es una linea del fichero referenciado.

Si se necesita leer contenido con acentos y caracteres especiales

- `open(fichero,'r',encoding='utf-8')` : Devuelve una lista de cadenas de caracteres donde cada cadena es una linea del fichero referenciado.

LECTURA DE DATOS

```
f = open('bienvenida.txt', 'r')
print(f.read())
f.close()
#¡Bienvenido a Python!
#¡Hasta pronto!

f = open('bienvenida.txt', 'r')
lineas = print(f.readlines())
print(lineas)
f.close()
['Bienvenido a Python!\n', '¡Hasta pronto!']
```

OPERACIONES CON FICHEROS

- Para renombra o borrar un fichero se utilizan funciones del módulo os:

`os.rename(ruta1, ruta2)` : Renombra y mueve el fichero de la ruta1 a la ruta2.

`os.remove(ruta)` : Borrael fichero de la ruta.

```
import os
f = 'bienvenida.txt'
if os.path.isfile(f):
    os.rename(f, 'saludo.txt') # renombrado
else:
    print('¡El fichero', f, 'no existe!')
```

OPERACIONES CON FICHEROS

- Antes de borrar o renombra un directorio conviene comprobar que existe para que no se produzca un error. Para ello se utiliza la función.
- `os.path.isfile(ruta)`: Devuelve True si existe un fichero en la ruta y False en caso contrario.

```
f = 'saludo.txt'
if os.path.isfile(f):
    os.remove(f) # borrado
else:
    print('¡El fichero', f, 'no existe!')
```

MÁS OPERACIONES

- `os.listdir(ruta)` : Devuelve una lista con los ficheros y directiorios contenidos en la ruta.
- `os.mkdir(ruta)` : Crea un nuevo directorio en la ruta.
- `os.chdir(ruta)` : Cambia el directorio actual al indicado por la ruta.
- `os.getcwd()` : Devuelve una cadena con la ruta del directorio actual.
- `os.rmdir(ruta)` :Borra el directorio de la ruta, siempre y cuando esté vacío.

LEER FICHERO DE INTERNET

- Para leer un fichero de internet hay que utilizar la función urlopen del módulo urllib.request.

urlopen(url): Abre el fichero con la url especificada y devuelve un objeto del tipo fichero al que se puede acceder con los métodos de lectura de ficheros anteriores.

```
from urllib import request
f=request.urlopen("https://firebasestorage.googleapis.com/
print(f.read())
```

LEER FICHERO DE INTERNET

- Al obtener un fichero que ha pasado a través de distintas redes con distintas codificaciones puedes obtener datos en formato con caracteres que producen errores.
- Original

```
Lorem Ipsum es simplemente el texto de relleno de las imprentas y
Lorem Ipsum ha sido el texto de relleno estándar de las industria
cuando un impresor (N. del T. persona que se dedica a la imprenta
galería de textos y los mezcló de tal manera que logró hacer un l
No sólo sobrevivió 500 años, sino que tambien ingresó como texto
quedando esencialmente igual al original. Fue popularizado en los
las cuales contenian pasajes de Lorem Ipsum, y más recientemente
el cual incluye versiones de Lorem Ipsum.
```

- Desde URL de internet

```
b'Lorem Ipsum es simplemente el texto de relleno de las imprentas y archivos de texto. \r\nLorem Ipsum ha sido el texto de relleno est\xc3\xa1ndar de las industrias desde
```

LEER FICHERO DE INTERNET

- El propio manejador de ficheros incluye una opcion para obtener un fichero en la codificacion deseada al leer datos

```
f=request.urlopen("https://firel
print(f.read().decode('utf-8'))
```

- Sin decode

```
b'MATRICULA;MODELO;PRECIO\r\n1234GPY;GT400;39500.65\r\n9845FEZ;IBIZA;10000.0\r\n4367HAB;MODEL S;105970.0\r\n4891UNO;F40;251000.0'
```

- Con decode

```
MATRICULA;MODELO;PRECIO
1234GPY;GT400;39500.65
9845FEZ;IBIZA;10000.0
4367HAB;MODEL S;105970.0
4891UNO;F40;251000.0
```

MANIPULAR DATOS CSV

- Python dispone de una librería estandar para el manejo de datos en formato CSV de manera rápida. Si estaís interesados (csv es un formato de intercambio bastante aceptado para hojas de cálculo de excel o similares) aquí teneis el enlace de documentación:

  https://docs.python.org/es/3/library/csv.html

MANEJO DE EXCEPCIONES

INTRODUCCION

- Python utiliza un objeto especial llamado excepción para controlar cualquier error que pueda ocurrir durante la ejecución de un programa.
- Cuando ocurre un error durante la ejecución de un programa, Python crea una excepción.
- Si no se controla esta excepción la ejecución del programa se detiene y se muestra el error (traceback).

```
>>> print(1 / 0)  # Error al intentar dividir por 0.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

TIPOS DE EXCEPCIONES

Los principales excepciones definidas en Python son:

- TypeError :Ocurre cuando se aplica una operación o función a un dato del tipo inapropiado.
- ZeroDivisionError : Ocurre cuando se itenta dividir por cero.
- OverflowError :Ocurre cuando un cálculo excede el límite para un tipo de dato numérico.
- IndexError : Ocurre cuando se intenta accedera una secuencia con un índice que no existe.

TIPOS DE EXCEPCIONES

- KeyError : Ocurre cuando se intenta accedera un diccionario con una clave que no existe.
- FileNotFoundError:Ocurre cuando se intenta acceder a un fichero que no existe en la ruta indicada.
- ImportError :Ocurre cuando falla la importación de un módulo.

CONTROL DE EXCEPCIONES

```
try:
    bloque código1
except excepción:
    bloque código2
else:
    bloque código3
```

CONTROL DE EXCEPCIONES

- Esta instrucción ejecuta el primer bloque de código y si se produce un error que genera una excepción del tipo especificado entonces ejecuta el segundo bloque de código.
- Mientras que si no se produce ningún error,se ejecuta el tercer bloque de código.

EJEMPLOS

```
def division(a, b):
    try:
        result = a / b
    except ZeroDivisionError:
        print('¡No se puede dividir por cero!')
    else:
        print(result)
```

EJEMPLOS

```
try:
    edad=int(input("Dime tu edad con números"))
except TypeError:
    print("No has escrito un número")
else:
    print("Gracias")
```

EJEMPLOS

```
try:
    f = open('fichero.txt') # El fichero no existe
except FileNotFoundError:
    print('¡El fichero no existe!')
else:
    print(f.read())
```

PROGRAMACIÓN ORIENTADA A OBJETOS

INTRODUCCIÓN

- Los objetos suelen representar conceptos del dominio del programa, como un estudiante, un coche, un teléfono, etc.
- Los datos que describen las características del objeto se llaman atributos y son la parte estática del objeto, mientras que las operaciones que puede realizar el objeto se llaman métodos y son la parte dinámica del objeto.

INTRODUCCIÓN

- Ejemplo. Una tarjeta de crédito puede representarse como un objeto:
  - ✓ Atributos: Número de la tarjeta, titular, balance, fecha de caducidad, pin, entidad emisora, estado (activa o no), etc.
  - ✓ Métodos: Activar, pagar, renovar, anular.

INTRODUCCIÓN

- Los objetos con los mismos atributos y métodos se agrupan clases. Las clases definen los atributos y los métodos, y por tanto, la semántica o comportamiento que tienen los objetos que pertenecen a esa clase.
- Se puede pensar en una clase como en un molde a partir del cuál se pueden crear objetos.
- Para declarar una clase se utiliza la palabra clave class seguida del nombre de la clase y dos puntos, de acuerdo a la siguiente sintaxis:

INTRODUCCIÓN

```
class <nombre-clase>:
    <atributos>
    <métodos>
```

- Los atributos se definen igual que las variables mientras que los métodos se definen igual que las funciones. Tanto unos como otros tienen que estar indentados por 4 espacios en el cuerpo de la clase.

INTRODUCCIÓN

- Para crear un objeto de una determinada clase se utiliza el nombre de la clase seguida de los parámetros necesarios.

```
clase(parámetros):
```

- Crea un objeto de la clase clase inicializado con los parámetros dados.
- Cuando se crea un objeto de una clase se dice que el objeto es una instancia de la clase.

INTRODUCCIÓN

- Los métodos de una clase son las funciones que definen el comportamiento de los objetos de esa clase.
- Se definen como las funciones con la palabra reservada def.
- La única diferencia es que su primer parámetro es especial y se denomina self.

INTRODUCCIÓN

- Este parámetro hace siempre referencia al objeto desde donde se llama el método, de manera que para acceder a los atributos o métodos de una clase en su propia definición se puede utilizar la sintaxis self.atributo o self.método.

INTRODUCCIÓN

- La razón por la que existe el parámetro self es porque Python traduce la llamada a un método de un objeto objeto.método(parámetros) en la llamada clase.método(objeto, parámetros), es decir, se llama al método definido en la clase del objeto, pasando como primer argumento el propio objeto, que se asocia al parámetro self.

MÉTODO __init__

- En la definición de una clase suele haber un método llamado __init__ que se conoce como inicializador.
- Este método es un método especial que se llama cada vez que se instancia una clase y sirve para inicializar el objeto que se crea.
- Este método crea los atributos que deben tener todos los objetos de la clase y por tanto contiene los parámetros necesarios para su creación, pero no devuelve nada.

MÉTODO __init__

- Se invoca cada vez que se instancia un objeto de esa clase.

```
class Tarjeta:
    def __init__(self, id, cantidad = 0):
        self._id = id
        self._saldo = cantidad
    def mostrar_saldo(self):
        print('El saldo es', self._saldo, '€')

# Creación de un objeto con argumentos
t = Tarjeta('1111111111', 1000)
t.muestra_saldo()
#El saldo es 1000 €
```

ATRIBUTOS DE INSTANCIA vs ATRIBUTOS DE CLASE

- Los atributos que se crean dentro del método __init__ se conocen como atributos del objeto, mientras que los que se crean fuera de él se conocen como atributos de la clase.
- Mientras que los primeros son propios de cada objeto y por tanto pueden tomar valores distintos, los valores de los atributos de la clase son los mismos para cualquier objeto de la clase.
- En general, no deben usarse atributos de clase, excepto para almacenar valores constantes o para tener valores que compartan todas las instancias de una misma clase.

ATRIBUTOS DE INSTANCIA vs ATRIBUTOS DE CLASE

- No existen modificadores de visibilidad (private o public), por lo que existen un acuerdo no escrito que si una variable empieza por _ significa que es una variable que no se pueden modificar desde fuera de la clase

```
class Circulo:
    # Atributo de clase
    pi = 3.14159
    def __init__(self, radio):
        # Atributo de instancia
        self._radio = radio
    def area(self):
        return Circulo.pi * self._radio ** 2
```

```
c1 = Circulo(2)
c2 = Circulo(3)
print(c1.area())
12.56636
print(c2.area())
28.27431
print(c1._pi)
3.14159
print(c2._pi)
3.14159
```

MÉTODO __str__

- Otro método especial es el método llamado __str__ que se invoca cada vez que se llama a las funciones print o str.
- Devuelve siempre una cadena que se suele utilizar para dar una descripción informal del objeto.
- Si no se define en la clase, cada vez que se llama a estas funciones con un objeto de la clase, se muestra por defecto la posición de memoria del objeto.

MÉTODO __str__

```
class Tarjeta:
    def __init__(self, numero, cantidad = 0):
        self._numero = numero
        self._saldo = cantidad
    def __str__(self):
        return f"Tarjeta número{self._numero} con saldo {self._saldo}€"

t = Tarjeta('0123456789', 1000)
print(t)
#Tarjeta número 0123456789 con saldo 1000.00€
```

MÉTODO __eq__

- Otro método especial es el método llamado __eq__ que es el equivalente al metodo equals de Java para poder comparar si dos objetos de la misma clase son iguales. Por ejemplo con la clase Tarjeta:

```
def __eq__(self,otra_tarjeta):
        return self.numero==otra_tarjeta.numero
```

HERENCIA

- Una de las características más potentes de la programación orientada a objetos es la herencia, que permite definir una especialización de una clase añadiendo nuevos atributos o métodos.
- La nueva clase se conoce como clase hija y hereda los atributos y métodos de la clase original que se conoce como clase madre.
- Para crear una clase a partir de otra existente se utiliza la misma sintaxis que para definir una clase, pero poniendo detrás del nombre de la clase entre paréntesis los nombres de las clases madre de las que hereda.

HERENCIA

- Ejemplo. A partir de la clase Tarjeta definida antes podemos crear mediante herencia otra clase Tarjeta_Descuento para representar las tarjetas de crédito que aplican un descuento sobre las compras.

```
class Tarjeta:
    def __init__(self, id, cantidad = 0):
        self.id = id
        self.saldo = cantidad
    # Método de la clase Tarjeta que hereda la clase Tarjeta_descuento
    def mostrar_saldo(self):
        print('El saldo es',  self.saldo, '€.')
```

HERENCIA

```
class Tarjeta_descuento(Tarjeta):
    def __init__(self, id, descuento, cantidad = 0):
        self.id = id
        self.descuento = descuento
        self.saldo = cantidad
    # Método exclusivo de la clase Tarjeta_descuento
    def mostrar_descuento(self):
        print('Descuento de', self.descuento, '% en los pagos.')
```

```
t = Tarjeta_descuento('0123456789', 2, 1000)
t.mostrar_saldo()
#El saldo es 1000 €.
t.mostrar_descuento()
#Descuento de 2 % en los pagos.
```

HERENCIA

- La principal ventaja de la herencia es que evita la repetición de código y por tanto los programas son más fáciles de mantener.
- En el ejemplo de la tarjeta de crédito, el método mostrar_saldo solo se define en la clase madre.
- De esta manera, cualquier cambio que se haga en el cuerpo del método en la clase madre, automáticamente se propaga a las clases hijas.

HERENCIA

- Sin la herencia, este método tendría que replicarse en cada una de las clases hijas y cada vez que se hiciese un cambio en él, habría que replicarlo también en las clases hijas.
- A partir de una clase derivada mediante herencia se pueden crear nuevas clases hijas aplicando de nuevo la herencia.
- Ello da lugar a una jerarquía de clases que puede representarse como un árbol donde cada clase hija se representa como una rama que sale de la clase madre.

HERENCIA

- Debido a la herencia, cualquier objeto creado a partir de una clase es una instancia de la clase, pero también lo es de las clases que son ancestros de esa clase en la jerarquía de clases.
- El siguiente comando permite averiguar si un objeto es instancia de una clase:

`isinstance(objeto, clase)`:

- Devuelve True si el objeto objeto es una instancia de la clase clase y False en caso contrario.

HERENCIA

```
t1 = Tarjeta('1111111111', 0)
t2 = t = Tarjeta_descuento('2222222222', 2, 1000)
isinstance(t1, Tarjeta)
True
isinstance(t1, Tarjeta_descuento)
False
isinstance(t2, Tarjeta_descuento)
True
isinstance(t2, Tarjeta)
True
```

SOBRECARGA Y POLIMORFISMO

- Los objetos de una clase hija heredan los atributos y métodos de la clase madre y, por tanto, a priori tienen tienen el mismo comportamiento que los objetos de la clase madre.
- Pero la clase hija puede definir nuevos atributos o métodos o reescribir los métodos de la clase madre de manera que sus objetos presenten un comportamiento distinto. Esto último se conoce como sobrecarga.

SOBRECARGA Y POLIMORFISMO

- De este modo, aunque un objeto de la clase hija y otro de la clase madre pueden tener un mismo método, al invocar ese método sobre el objeto de la clase hija, el comportamiento puede ser distinto a cuando se invoca ese mismo método sobre el objeto de la clase madre.
- Esto se conoce como polimorfismo y es otra de las características de la programación orientada a objetos.

SOBRECARGA Y POLIMORFISMO

```
class Tarjeta:
    def __init__(self, id, cantidad = 0):
        self.id = id
        self.saldo = cantidad
    def mostrar_saldo(self):
        print('El saldo es {:.2f}€.'.format(self.saldo))
    def pagar(self, cantidad):
        self.saldo -= cantidad
class Tarjeta_Oro(Tarjeta):
    def __init__(self, id, descuento, cantidad = 0):
        self.id = id
        self.descuento = descuento
        self.saldo = cantidad
    def pagar(self, cantidad):
        self.saldo -= cantidad * (1 - self.descuento / 100)
```

SOBRECARGA Y POLIMORFISMO

```
t1 = Tarjeta('1111111111', 1000)
t2 = Tarjeta_Oro('2222222222', 1, 1000)
t1.pagar(100)
t1.mostrar_saldo()
#El saldo es 900.00€.
t2.pagar(100)
t2.mostrar_saldo()
#El saldo es 901.00€.
```

SUPER

- Si queremos usar en una clase Hija cualquier método de una clase tenemos que usar la llamada super() como Java:

```
class Tarjeta_Oro(Tarjeta):
    def __init__(self, id, descuento, cantidad = 0):
        super().__init__(id,cantidad)
        self.descuento = descuento
```

Bibliografía y enlaces

APUNTES MANUALES Y TUTORIALES:

- Autor original: Guido van Rossum, Fred L. Drake, Jr “El tutorial de Python”. Python Software Foundation 2018
- Documentación de usuario de Odoo https://www.odoo.com/documentation/user/15.0/es/
- Documentación del desarrollador de Odoo https://www.odoo.com/documentation/15.0/es/

VIDEO-TUTORIALES:

- Python For Everybody https://www.youtube.com/watch?v=o0XbHvKxw7Y
- TechWorld with Nana https://www.youtube.com/watch?v=t8pPdKYpowI
- Freecodecamp https://www.youtube.com/watch?v=DLikpfc64cA
- Píldoras informáticas https://www.youtube.com/watch?v=G2FCfQj-9ig&list=PLU8oAlHdN5BlvPxziopYZRd55pdqFwkeS

FIN DEL TEMA 4!

Cualquier duda enviar un correo a
daniel.lopez@escuelaartegranada.com