MINI APUNTES PYTHON

Sistemas de Gestión Empresarial

13 DE NOVIEMBRE DE 2024

ERIK AMO TOQUERO

Contenido

Introducción

Python es un lenguaje de programación muy conocido por el hecho de que es muy anárquico. No utiliza indentaciones de ningún tipo, una variable puede cambiar su tipo cuantas veces quiera... Aún así a las empresas les gusta mucho su versatilidad, por lo que se utiliza mucho.

Tipos de variables

Las variables en Python no son muy importantes que se diga, ya que su tipo de dato puede cambiar de un momento a otro. No obstante, es necesario saber qué hace cada tipo:

- **int:** Entero
- **float:** Decimal de coma flotante
- **double:** Decimal
- **str:** Cadena de caracteres
- **bool:** Booleana
- **char:** Carácter

Estructuras de Control

Como con todos los lenguajes, una primera toma de contacto es el uso de estructuras de control

Salida de Información

Se utiliza **print(String, end)**, siendo end un atributo opcional que marca cómo termina el print (Si hay un salto de línea o no, por ejemplo).

```
print(f"Tu número es {numero}")#f añade formato al texto
```

Entrada de Información

Se utiliza input(str). Ese string es uno que salta en consola para pedir que introduzca un valor. Input devuelve *str*, pero se puede parsear.

Condicionales

Condicional Simple

Igual que en Java:

```
if(condicion):
        [...] # IMPORTANTE LA TABULACIÓN
```

Condicional Doble

Simple, un if-else

```
if(condicion):
        [...]
    else:
        [...]
```

If-Else-If

Manejo de varias condiciones. Else if se puede escribir como elif

```
if condicion1:
        [...]
    elif condicion2:
        [...]
    else:
        [...]
```

Match – El switch de Python

En este no aparece el caso default, sino que aparece **case** _

```
match (num):
        case 1: [...]
        case 2: [...]
        case _: [...]
```

Operador Ternario – If-Else resumido

En Python sigue una lógica muy extraña comparada con el resto: (valor falso, valorverdadero)[condición]

```
condicion1:bool = False
    num1 = (2, 1)[condicion1]
    print(num1) #Saca 2
```

Bucles – Estructuras iterativas

While – Bucle condicional

Hasta que no se cumpla una condición, no termina el bucle

```
while(condicion){
    [...]
}
```

For

El bucle for en Python tiene varias formas de hacerse y recorrerse:

- **For de Rango X..Y**: Recorre desde el primer número hasta el último

```
for i in range(1, 9):
    print(i) #Imprime todos los numeros de 1 a 9
```

- For de Conjuntos: Recorre todos los objetos de un conjunto

```
for i in strings:
    print(i) #Imprime Hola, luego Adios, luego Buenas
```

- For de Strings: Lee carácter por carácter un String

```
for i in "Hola":
    print(i) #Imprime primero H, luego o, luego l, y luego a
```

Funciones

Hay dos tipos de funciones en Python: Las definidas y las lambdas.

Funciones Definidas

Son aquellas que se guardan en lo que es una función. Se declaran con def nombrefunción(parámetros):[Valor que devuelven]

```
def funcion(i:int)->str:
    [...]
    return string
```

Funciones Lambda

Son funciones anónimas que se guardan en variables. Esto es útil a la hora de querer ahorrar código y crear las funciones de orden superior (Las que usan como parámetro otra función)

```
y = lambda a, b, c : a + b + c #Función que devuelve la suma de 3
parámetros que se le pasa
print(y(1, 2, 3))
#Función que devuelve una lambda ("Recursión")
def funcion(n):
    return lambda a : a + n #Lambda coge n de la función y la suma a otro
valor recibido por paréntesis

print(funcion(2)(11)) #El primer paréntesis es n, y el segundo es a

def ordenSuperior(funcion, numero):
    if(funcion(numero)):
        print("Si")
ordenSuperior(lambda n:n%2==0, 6)
```

Estructuras de Datos – Colecciones

Las colecciones son conjuntos de datos hechos para guardar distintas variables en una estructura dinámica.

Lista

Son almacenes de objetos ordenados por índices (empezando por 0). No tiene la necesidad de otros lenguajes de declarar un tipo para la lista, ya que puede guardar de todo en todo momento. No obstante, si queremos usar la lista para un único tipo, no está de menos

Creación de listas: Declarando un tipo y un valor fijo de índices, o simplemente meter datos entre corchetes separados por comas

```
nuevaLista:int[12] #Declaración de una lista con valor fijo de tipo int
lista = ["Obj1", "Paco", 23, "Hola", "Genial", "Pedro"]
```

Acceso al contenido de la lista:

```
print(lista[0]) #Se refiere al primer item
print(lista[-1]) #Se refiere al último item
print(lista[2:4]) #Muestra el rango entre 2 y 4 SIN INCLUIR EL 4
print(lista[:2]) #Muestra hasta la posición 2 (0, 1, 2)
print(lista[2:]) #Muestra desde la posición 2 hasta el final
boolean = ("Hola" in lista) #'' in list es una booleana que mira si un objeto está en la lista
```

Cambio de objetos en las listas: Se puede hacer por rango o por item

```
lista[1:3] = ["Merlina", "Jose"]#Cambias del rango 1 a 3 SIN INCLUIR EL 3
```

Inserción: Hay 3 formas: insert(índice, objeto), append(objeto), y extend(conjunto)

```
lista.insert(1, "AAAA") #Mueve todo una posición a la derecha para meter en el puesto 1 el objeto
lista.append(3.141592) #Inserta al final
lista.extend(nuevaLista) #Inserta una lista al final de la otra (en este caso nuevaLista al final de lista)
```

Recorrer listas: Hay 4 formas: ForEach, For con longitud, while con incremento y realizar una acción con un for:

```
for x in lista:
    print(x) #ForEach
for i in range(len(lista)):
    print(lista[i]) #For con incremento que recorre la lista
i=0
while(i<len(lista)):
    print(lista[i])
    i+=1 #While que recorre la lista

[print(x) for x in lista] #Imprime con un for
```

Ordenar: Hay 3 formas: sort, sort(reverse) y reverse

```
lista.sort() #Sort de la lista sin más
lista.sort(reverse=True) #Sort descendiente
lista.reverse #Sort al revés
```

- **IMPORTANTE:** Los shorts de palabras van de la siguiente forma:
  1. Números
  2. Letras de la A a la Z en mayúscula
  3. Letras de la a a la z en minúscula

Copiar

```
lista2 = lista.copy() #Sin más
lista3 = lista.list(lista) #Otro tipo de copia
lista4 = lista[:] #Copia cada elemento de la lista
```

Unir

```
lista5 = lista2+lista3 #"Suma" de listas
lista6 = [1, 2, 3]
for x in lista:
    lista6.append(x) #Va añadiendo al final cada elemento.
lista7=[3, 5, 10]
lista7.extend(lista6) #Añade la lista 6 al final de la 7.
```

Métodos de Listas

```
lista7.clear() #Borra los elementos de la lista
lista.count() #Muestra cuantos elementos tiene la lista
lista.index("Hola") #Muestra el índice del elemento en la lista.
```

Eliminar objetos

```
lista.remove("Genial") #Borra el nombre si lo encuentra. Si hay varias
ocurrencias, borra la primera
lista.pop(0) #Borra según el índice
lista.pop() #Borra el último elemento
del lista[2] #Pop 2.0
del lista #Borra la lista completamente
```

Set

Son listas sin orden pero con una condición especial: No se pueden repetir valores.

Comparte muchos de los comandos de las listas, variando en lo siguiente:

Actualización: No existe, ya que no se puede acceder al contenido del set de forma normal

Creación: Solo se puede hacer así:

```
set = {"Perro", "Hola", 123}
```

Adición: No añadirá nada si ya está ese objeto en el set

```
set.add("Adios")
```

Eliminación: Borrará el objeto si lo encuentra

```
set.remove("Perro")
```

Unión de sets

```
set.union(set2) #Une sets. Permite meter unos cuantos de una
```

Tupla

Hablando rápidamente, las tuplas son unas listas con muchas (pero muchísimas) restricciones, ya que solo se pueden meter valores al momento de su declaración.

Los pasos para hacer casi todo en la tupla son convertir a lista, hacer la operación y pasar el resultado a tupla. Salvo los siguientes:

Creación

```
tupla = ("Hola", "adios", 19, "asdf") # Tupla con muchos items
tupla2 = ("a",) #Tupla de un solo objeto -> Sin la coma sería un objeto
normal
tupla3 = tuple(("Hola", "Adios", "Perro")) #tuple es el Constructor
```

Acceso a tuplas: Como en las listas la verdad

```
print(tupla[0]) #Muestra el valor del indice 0 de la tupla
print(tupla[-1]) #Muestra el último valor
print(tupla[1:3]) #Muestra la tupla entre los índices 1 y 3 SIN INCLUIR
EL 3
print(tupla[:2]) #Muestra del principio a la posición 2 de la tupla
print(tupla[1:]) #Muestra del valor 1 al final de la tupla
```

Unión de tuplas

```
tupla = tupla + tupla2 #Esto sí se permite por algún motivo
```

Cuanto más sabes de Python, más te cuestionas si de verdad sabes de Python

Desempaquetado: Dar valores de una tupla a objetos

```
nuevaTupla = tuple(("Hola", "Buenas", "Adios"))
(prueba1, prueba2, prueba3) = nuevaTupla #Asigno el primer valor al
indice 0, el segundo al indice 1 y así
(prueba1, *prueba2) = nuevaTupla #Si hay más valores en la tupla que en
la toma de valores, se puede añadir un * para que el valor se convierta
en una lista
(*prueba2, prueba3) = nuevaTupla #Ese asterisco se puede poner donde
quiera, cabe recalcar
```

Multiplicación

```
nuevaTupla = nuevaTupla*2 #Si, por algún motivo puedes hacer esto
```

(Nota del escritor: Cada día me hierve más la sangre cuando veo esto)

Diccionario

Los diccionarios guardan información en pares clave-valor. De estos diccionarios luego puede salir un archivo JSON o al revés.

Declaración: Literalmente ponerlo en formato JSON:

```
diccionario = {"Hola":"Adios",
               "24":"Hello",
               "Año": 2024
               }
```

Acceso a los datos

```
print(diccionario) #Imprime todo
print(diccionario["Año"]) #Si pones la clave te da el valor
#print(diccionario[2024]) #Esto no funciona
print(diccionario.get("Año")) #Funciona lo mismo
print(diccionario.keys()) #Imprime las claves del diccionario
print(diccionario.values()) #Imprime los valores del diccionario
tupla = diccionario.items() #Devuelve los items del diccionario a modo de
tupla
boolean = ("Año" in diccionario) #Guarda si existe la CLAVE
```

Añadir contenido

```
diccionario["ASDF"] = "A" #Si metes una clave inexistente, la crea
diccionario.update({"Aa" : "Hi"}) #Igual con el update
```

Actualizar contenido

```
diccionario["Año"] = 2025 #Cambia el valor si existe la clave
diccionario.update({"Año" : 2024}) #Actualiza el valor de x clave si
existe
```

Bucles: Es bastante parecido a las listas

```
for x in diccionario:
    print(x) #ForEach de las claves
for x in diccionario:
    print(diccionario[x]) #ForEach de valores
for x in diccionario.values():
    print(x) #ForEach de valores 2.0
for x in diccionario.keys():
    print(x) #ForEach de claves 2.0
for x, y in diccionario.items():
    print(f"Clave: {x} ; valor: {y}") #x es clave e y es valor
```

Borrar contenido: Parecido también a las listas

```
diccionario.pop("ASDF") #Borra la clave
diccionario.popitem() #Borra la última clave metida
del diccionario["24"] #Borra esa clave
del diccionario #Borra el diccionario
```

Manejo de Ficheros

En Python, una librería es un conjunto de módulos que contienen funciones, clases y variables para reutilizar

Las librerías python permiten aprovechar el código de otros desarrolladores que aceleran el desarrollo de programas.

- Los módulos son archivos que contienen el código Python (.py)
- Las librerías son un conjunto de esos módulos relacionados entre sí

Hay varios tipos de librerías:

- Estándar: Vienen incluidas en Python (math, os, datetime, random…)
- Externas: Van aparte y se tienen que importar en el workspace (NumPy, Panda, request…)

El manejo de ficheros se hace con la librería 'os' para crear y/o borrar y usa el comando open(archivo, 'w' (sobreescribir) / 'r' (leer) / 'a' (continuar)) para ver o editar su contenido

```
import os #Se importa la librería de manejo de archivos
file_name = "fichero.txt" #El nombre del fichero es el lugar en el que se encuentra
desde la raíz del proyecto + su nombre
#Si no encuentra un archivo, lo crea.
with open(file_name, "w") as file: #Abro el fichero en modo escritura "w" = "write"
    file.write("Prueba\n")
    file.write("Nombre\n")
    file.write("a\n")
with open(file_name, "r") as file: #Abro el fichero en modo lectura "r" = "read"
    print(file.readline())
    print(file.readline())
    print(file.readline())
with open(file_name, "w") as file:
    file.flush() #File.flush() sirve para borrar todo el contenido del fichero
os.remove(file_name) #Con esto se elimina el archivo
```

Clases – POO con Python

Python es un lenguaje de Programación Orientado a Objetos, es decir, puede crear objetos para realizar distintas acciones y métodos con cada uno de ellos

La creación de una clase se hace con class nombre: y ya dentro todos sus atributos y métodos

Por defecto, el constructor principal de las clases es el constructor vacío, pero se puede crear más constructores con el método __init__(objeto, valores)

```
class Animal:
    name:str
    def __init__(self, name):
        self.name = name
    def sound(self):
        pass
```

La creación de un objeto se haría de la siguiente forma:

```
my_animal = Animal("Animal") #Recibe el nombre como parámetro para el constructor
```

Herencia, polimorfismo y sobrecarga

Esto es igual a todos los lenguajes de POO:

- La herencia (abstracción) es el paso de métodos y atributos de una clase padre a sus hijos
- El polimorfismo es la capacidad de poder introducir en un objeto de tipo clase padre alguno de sus hijos (ahora se verá un ejemplo más visual)
- La sobrecarga es la capacidad de poder cambiar una variable o función recibida de una superclase para adaptarla al gusto de cada subclase

```
class Dog(Animal): #Declaración de clase hija: class hija(padre)
    def sound(self): #Sobrecarga de método sound
        print("Guau")
class Cat(Animal):
    def sound(self):
        print("Miau")
my_dog:Animal = Dog("Perro") #Polimorfismo
my_dog.sound() #Llamada de método
my_cat = Cat("Gato")
my_cat.sound() #Llamada de método
```

Manejo de Excepciones

Cuando hay un error inesperado en un programa y este no sabe solucionarlo, salta una excepción que hace que se cierre.

El control de excepciones busca, por medio de try-except-finally, manejar esa excepción, continuando el flujo del programa si salta el error.

```
try: #Inicio de la búsqueda de excepciones
    numero:int = int(input("Escribe un número: ")) #Si se escribe una
letra, salta una excepcion
    print("Tu numero es " + numero) #Esto solo sale si no salta la
excepcion
except: #Salto al control de excepciones
    print("No es un número")
finally: #Da igual si salta o no la excepción, saltará lo siguiente al
finalizar.
    print("Se ha terminado el control de la excepción")
```

JSON

JSON es un tipo de fichero cuyo nombre completo es JavaScript Object Notation, dígase es la forma en la que se guardan los objetos de JavaScript. Estos ficheros son interoperables con otros lenguajes de programación por el hecho de que no dejan de ser pares clave-valor.

La librería json permite controlar un json como si fuera un diccionario de Python, pudiendo escribir luego esa información en un archivo.

```python
import json #Es importante importar la librería json para poder abrirlo
import os
x = '{ "nombre":"Pedro", "edad":19, "ciudad":"Murcia"}' #Forma de JSON
y = json.loads(x) #Parsea el JSON
print(y) #Imprime el JSON entero
print(y["nombre"]) #Imprime solo el campo del nombre
#Convertir Python a JSON
x = { #Diccionaro en Python
    "nombre": "Jose",
    "edad": 20,
    "ciudad": "Madrid"
}
y = json.dumps(x) #Convierte el diccionario a JSON
print(y) #Escribe el JSON
with open('alumnos.json', 'w') as outfile: #Abre el archivo erik.json en modo escritura
    json.dump(x, outfile) #Mete en el archivo el json (outfile) el contenido
#El manejo de archivos json permite poder meter más de un dato en el diccionario o en el archivo
dictAlumnos = []
with open('alumnos.json', 'r') as infile: #Lee el json con más de un objeto
    dictAlumnos = json.load(infile) #Mete en el diccionario el contenido del json bien estructurado
print(dictAlumnos)
for x in dictAlumnos:
    print(x["nombre"])
alumno = {
    "Nombre" : "Mario",
    "edad" : 18,
    "Fecha de nacimiento" : "04-01-2005",
    "Módulos" : []
}
dictAlumnos = []
dictAlumnos.append(alumno)
alumno = {
    "Nombre" : "Angel",
    "edad" : 19,
    "Fecha de nacimiento" : "15-06-2005",
    "Módulos" : ["Programación Multimedia y Dispositivos Móviles", "Sistemas de Gestión Empresarial", "Programación de Servicios y Procesos", "Acceso a Datos", "Desarrollo de Interfaces"]
}
dictAlumnos.append(alumno)
with open('alumnos.json', 'w') as outfile:
    json.dump(dictAlumnos, outfile)

with open('alumnos.json', 'r') as infile:
    print(infile.read())
os.remove('alumnos.json') #Borra el archivo
```

HTTP

Los servicios Web son aplicaciones que permiten la comunicación con distintos dispositivos electrónicos en red. Esto se hace por el protocolo HTTP: Un protocolo de transferencia de archivos de hipertexto por la red. Se basa en que el cliente hace una **petición** (un mensaje para hacer alguna instrucción en red) al servidor, y este devuelve una respuesta en forma de código:

200: Aceptado
400: Petición no encontrada
500: Error interno

Python controla todo esto con la librería requests, la cual permite hacer peticiones HTTP y recibir respuestas.

```
import requests
x = requests.get('https://www.google.com/')
print(x.status_code) #Imprime el código de la respuesta
if x.status_code != 200:
    print("Error de petición", x.status_code)
else:
    print(x.text)  # Imprime el HTML en este caso de la web
```

APIs y RESTful APIs

Un API es una especie de contrato que permite a los desarrolladores interactuar con una aplicación por distintas interfaces.
Una RESTful API es una API que se ajusta al uso del desarrollador. Estas se forman de distintos EndPoints: puntos donde interaccionan el cliente y servidor, donde se recibe la información del servidor (si el código de respuesta es 200)
Las RESTful APIs devuelven la información en formato JSON, por lo que es necesario parsearlo una vez se saca

Para el ejemplo, voy a usar la PokeAPI, una RESTful API que permite sacar toda la información de los Pokémon:

```
import requests
statusCode = 200
i = 1
while statusCode == 200:
    url = f"https://pokeapi.co/api/v2/pokemon/{i}/" #Accedo a la API
    response = requests.get(url) # Petición GET
    statusCode = response.status_code #Codigo de respuesta
    if statusCode == 200: #Si se ha conectado:
        data = response.json() #Transformo a diccionario los datos
        print(f"{data["name"].capitalize()}", end=", ") #Imprimo el
nombre en formato "nombre, "
    i+=1 #Sumo 1 al contador
```