Python Avanzado

Tecnofor
by SNGULAR

Contenidos

- **Python Avanzado**

  - Unidad 1: Manejo de Excepciones
  - Unidad 2: Programación Funcional y Concurrencia
  - Unidad 3: Decoradores y Generadores
  - Unidad 4: Librerías para Análisis y Visualización de Datos
  - Unidad 5: Introducción al Machine Learning con Scikit-learn
  - Unidad 6: Introducción a Numpy

Unidad 1: Manejo de Excepciones

TecnoFor by SNGULAR

try-except

```
try:
    print(3/0)
except:
    print("ERROR: Division por cero")
```

try-except-finally

```
print("¡Iniciando programa!")
try:
    print(3/0)
except:
    print("ERROR: Division erronea")
finally:
    print("¡Programa acabado!")
```

try-except-else-finally

```
print("¡Iniciando programa!")
try:
    print(3/1)
except:
    print("ERROR: Division erronea")
else:
    print("¡No se han producido errores!")
finally:
    print("¡Programa acabado!")
```

Captura varias excepciones

```
print("¡Iniciando programa!")
try:
    print(3/0)
except ZeroDivisionError:
    print("ERROR: Division por cero")
except:
    print("ERROR: General")
else:
    print("¡No se han producido errores!")
finally:
    print("¡Programa acabado!")
```

Excepciones habituales

- **ZeroDivisonError**: Esto aparece cuando intenta forzar a Python a realizar cualquier operación que provoque una división en la que el divisor es cero o no se puede distinguir de cero. Tenga en cuenta que hay más de un operador de Python que puede provocar que surja esta excepción. /, // o %

- **ValueError**: Espere esta excepción cuando esté tratando con valores que pueden usarse de manera inapropiada en algún contexto. En general, esta excepción surge cuando una función (como int() o float()) recibe un argumento de un tipo adecuado, pero su valor es inaceptable.

- **TypeError**: Esta excepción aparece cuando intenta aplicar un dato cuyo tipo no se puede aceptar en el contexto actual.

- **AttributeError**: Esta excepción llega -entre otras ocasiones- cuando intentas activar un método que no existe en un elemento con el que estás tratando.

- **SyntaxError**: Esta excepción se produce cuando el control llega a una línea de código que viola la gramática de Python.

Unidad 2: Programación Funcional y Concurrencia

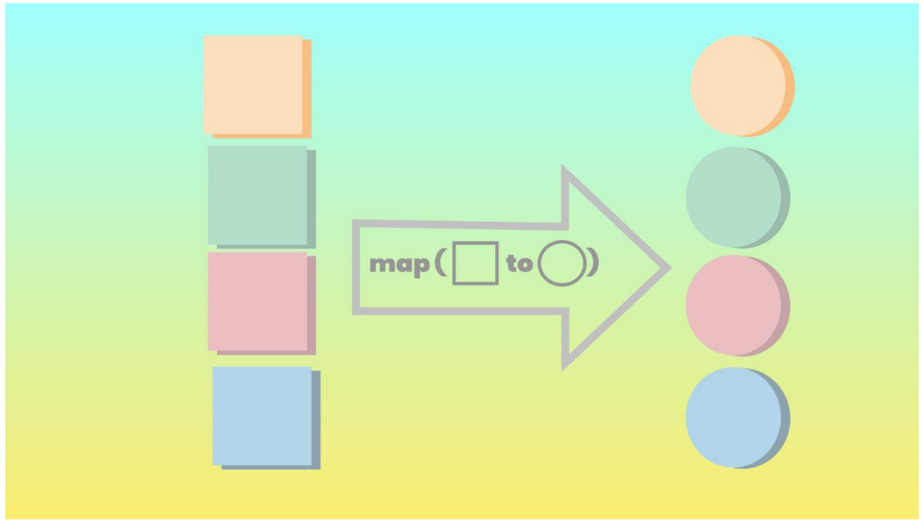
TecnoFor by SNGULAR

Funciones anidadas

```
def SumarRestar(param1, param2):
    return Sumar(param1,param2), Restar(param1,param2)

def Sumar(sumando1, sumando2):
    return sumando1 + sumando2

def Restar(minuendo, sustraendo):
    return minuendo - sustraendo

numero1 = int(input("Introduce el primer numero: "))
numero2 = int(input("Introduce el segundo numero: "))
resultadosuma, resultadoresta = SumarRestar(numero1,numero2)
print("El resultado de la suma es: ", resultadosuma)
print("El resultado de la resta es: ", resultadoresta)
```

Función map

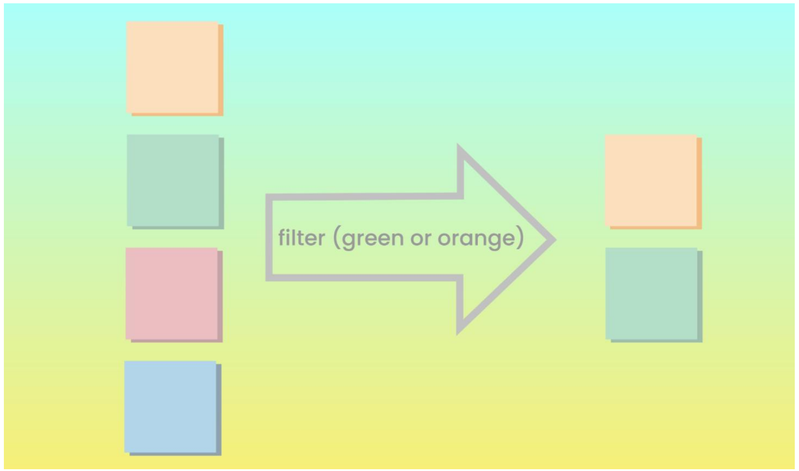
- La función map nos permite aplicar una función sobre cada uno de los elementos de un colección (Listas, tuplas, etc...).




```
map(función a aplicar, objeto iterable)
```

Función filter

- La función filter, es quizás, una de las funciones más utilizadas al momento de trabajar con colecciones. Cómo su nombre lo indica, esta función nos permite realizar un filtro sobre los elementos de la colección.




```
filter(función a aplicar, objeto iterable)
```

Función reduce

- Usaremos la función reduce cuando poseamos una colección de elementos y necesitemos generar un único resultado. reduce nos permitirá reducir los elementos de la colección. Podemos ver a esta función como un acumulador.

```
reduce(función a aplicar, objeto iterable)
```

Funciones lambda

- Una función lambda es una función anónima, una función que no posee un nombre. En Python la estructura de una función lambda es la siguiente.

```
lambda argumentos : cuerpo de la función
```

Introducción a la concurrencia

- La concurrencia en Python es la capacidad de un programa para ejecutar múltiples tareas de manera que parezcan simultáneas, aunque no necesariamente se ejecuten exactamente al mismo tiempo.

- Esto permite que un programa sea más eficiente al aprovechar mejor los recursos disponibles y mejorar la capacidad de respuesta, especialmente en aplicaciones que realizan muchas operaciones de entrada/salida (E/S) o que requieren manejar múltiples tareas al mismo tiempo

Diferencia entre concurrencia y paralelismo

- **Concurrencia**: Varias tareas avanzan de manera intercalada, compartiendo recursos, pero no necesariamente ejecutándose al mismo tiempo.
- **Paralelismo**: Varias tareas se ejecutan literalmente al mismo tiempo, por ejemplo, en diferentes núcleos de CPU

Herramientas de concurrencia

- Python ofrece varias herramientas y módulos para implementar concurrencia, cada una adecuada para diferentes tipos de tareas:

Herramienta	Descripción	Uso principal
`threading`	Permite ejecutar múltiples hilos dentro de un mismo proceso, compartiendo memoria	Tareas de E/S, operaciones ligeras
`multiprocessing`	Ejecuta procesos independientes, cada uno con su propia memoria	Tareas intensivas en CPU
`concurrent.futures`	Proporciona una interfaz de alto nivel para ejecutar tareas concurrentes con hilos o procesos	Simplificar la administración de tareas
`asyncio`	Permite programación asíncrona basada en corrutinas y bucle de eventos	E/S asíncrona, servidores web
`subprocess`	Ejecuta procesos externos y permite la comunicación con ellos	Automatización de comandos del sistema

Unidad 3: Decoradores y Generadores

TecnoFor by SNGULAR

Decoradores

- Un decorador es uno de los patrones de diseño que describe la estructura de objetos relacionados. Python es capaz de decorar funciones, métodos y clases.

- Los decoradores se utilizan en:
  - la validación de argumentos
  - la modificación de argumentos
  - la modificación de los objetos devueltos;
  - la medición del tiempo de ejecución
  - el registro de mensajes
  - la sincronización de hilos;
  - la refactorización del código;
  - almacenamiento en caché.

Decoradores

- Sin decorador

```
def simple_hello():
    print("Hello from simple function!")
def simple_decorator(function):
    print('We are about to call "{}"'.format(function.__name__))
    return function
decorated = simple_decorator(simple_hello)
decorated()
```

- Con decorador

```
def simple_decorator(function):
    print('We are about to call "{}"'.format(function.__name__))
    return function


@simple_decorator
def simple_hello():
    print("Hello from simple function!")


simple_hello()
```

Generadores e iteradores

- Un generador de Python es un fragmento de código especializado capaz de producir una serie de valores y controlar el proceso de iteración.

- La función range() es un generador, la cual también es un iterador.

```
for i in range(5):
    print(i)
```

Iterador

- El protocolo iterador es una forma en que un objeto debe comportarse para ajustarse a las reglas impuestas por el contexto de las sentencias for e in.

- Un iterador debe proporcionar dos métodos:
  - __iter__() el cual debe devolver el objeto en sí y que se invoca una vez (es necesario para que Python inicie con éxito la iteración).
  - __next__() el cual debe devolver el siguiente valor (primero, segundo, etc.) de la serie deseada: será invocado por las sentencias for/in para pasar a la siguiente iteración; si no hay más valores a proporcionar, el método deberá generar la excepción StopIteration.

- yield en lugar de return. Esta pequeña enmienda convierte la función en un generador

```
def fun(n):
    for i in range(n):
        yield i


for v in fun(5):
    print(v)
```

Generadores con listas de compresión

- Los generadores también se pueden usar con listas por comprensión

```
def powers_of_2(n):
    power = 1
    for i in range(n):
        yield power
        power *= 2


t = [x for x in powers_of_2(5)]
print(t)
```

- Los corchetes hacen una comprensión, los paréntesis hacen un generador.

```
the_list = [1 if x % 2 == 0 else 0 for x in range(10)]
the_generator = (1 if x % 2 == 0 else 0 for x in range(10))

for v in the_list:
    print(v, end=" ")
print()

for v in the_generator:
    print(v, end=" ")
print()
```

Unidad 4: Librerías para Análisis y Visualización de Datos

TecnoFor by SNGULAR

Módulo Pandas

- Esta librería es fundamental en el tratamiento de datos.
- Nos permite filtrar datos, manipularlos, transformarlos, etc.
- Podemos leer y escribir diferentes tipos de ficheros como csv y excel. Tambien en BBDD.
- Veremos dos de las tres estructuras que tiene Pandas:
    - Series
    - Data Frames

Series

- Características:
  - Son tipo vectores o arrays de una dimension
  - Todos los datos tienen que ser del mismo tipo
  - Contiene indices, es decir, un elemento de la serie puede asociarse con ese indice. Esto es muy util a la hora de acceder a los datos

- Constructor para crear una serie:

```
Series(data, index, dtype)
```

donde data puede ser por ejemplo una lista o un diccionario.

Series

- Atributos de las series:
  - size: nos da la longitud de la serie
  - index: devuelve los indices de la serie

```
# size
print('La longitud de la serie es: ')
print(serieDesdeDict.size, '\n')

# index
print('Los indices de la serie son: ')
print(serieDesdeDict.index, '\n')
```

Series

- Acceder a los elementos de una serie
  - Por posición:

```
#Ejemplo 1: quiero los dos primeros elementos
print('Ejemplo 1: \n ', serieDesdeDict[:2], '\n')

#Ejemplo 2: quiero los tres ultimos elementos
print('Ejemplo 2: \n ', serieDesdeDict[-1])
```

  - Por indice

```
#Ejemplo 3: quiero el elemento cuyo indice es el 0
print('Ejemplo 3: \n ', serieDesdeDict[0], '\n')

#Ejemplo 4: quiero los elementos cuyos indices son el 1 y el 4
print('Ejemplo 4: \n ', serieDesdeDict[[1, 4]])
```

Series

- Método apply junto con funciones lambas:
  - La función apply() aplica una función a cada elemento a lo largo de la serie.
  - La sintaxis es la siguiente:

```
nuestraserie.apply(Nuestrafuncion)
```

- La ventaja de apply es que nos evitamos usar los bucles, que realmente son más lentos, sobre todo con datos muy grandes.

```
serieFiltro = serieDesdeDict.apply(lambda x: x in ['Pedro', 'Juan'])
print(serieFiltro)
```

Data Frames

- Características:
  - Son tablas, es decir, son de 2 dimensiones.
  - Cada columna es una serie.
  - Contiene dos índices, uno columnar y otro por fila como las series. Y estos deben ser únicos.

- NOTA: Como las series y los data frames pertenecen a la misma clase, todo lo visto anteriormente se aplica a los data frames.

Data Frames

- Creación de un Data Frame:
  - Constructor para crear un data frame:

```
DataFrame(data, index, columns, dtype)
```

donde data puede ser por ejemplo una lista o un diccionario.

  - Para crear un data frame vacio

```
df = pd.DataFrame()
```

Data Frames

- Data frame a partir de una lista:

```
dfDesdeLista = pd.DataFrame(data)
```

- A partir de una lista de lista:

```
data2 = [['Juan', 4.5], ['Pedro', 8.9], ['Estefania', 1.4], ['Ana', 5.6], ['Esteban', 7.8]]
columnasNombres = ["Nombre", "Notas"]
filas = list(range(5))

dfDesdeListDeLista = pd.DataFrame(data2, columns=columnasNombres, index=filas)
```

- A partir de un diccionario:

```
dictNombreNotas1 = { "Id":[1,2,3,4,5] ,
    "Nombre" : ['Juan', 'Pedro', 'Estefania', 'Ana', 'Esteban'] ,
    "Apellido":['Garcia', 'Sanchez', 'Lopez', 'Garcia', 'Gonzalez' ],
    "Notas": [4.5, 8.9, 1.4, 5.6, 7.8]}

dfDesdeDict1 = pd.DataFrame(dictNombreNotas1)
```

- A partir de la lectura de un csv:

```
dfDatosPersonales = pd.read_csv('C:/DatosPersonales.csv',sep=";")
```

Data Frames

- Atributos de los data frames:
  - info: nos devuelve toda la información referente al data frame
  - shape: nos devuelve el número de filas y columnas del data frame
  - size: devuelve el número de elementos en el data frame
  - columns: nos da una lista con el nombre de las columnas
  - index: nos devuelve una lista con el nombre de los índices fila
  - head(n): nos da los n primeros elementos del data frame

```
print('Informacion del df: ', dfDesdeDict1.info, '\n')

print('Numero de filas y columnas que tiene: ', dfDesdeDict1.shape, '\n')

print('Numero total de datos: ', dfDesdeDict1.size, '\n')

print('Nombre de las columnas: ', dfDesdeDict1.columns, '\n')

print('Nombre de los indices: ', dfDesdeDict1.index, '\n')

print('Las primeras dos lines: \n')
print(dfDesdeDict1.head(2))
```

Data Frames

- Acceder a los datos:
  - Por posicion se usa "iloc"
  - Por nombre del indice fila es traves de "loc"
  - Por nombre de columna es df[nombre de la columna] o la lista con los nombres de las columnas si queremos varias a la vez

Nota: Recordar que en Python los indices comienzan desde 0

```
#Queremos obtener el dato de la columna 1 y fila 3
dfDesdeDict1.iloc[3,1]
```

```
#Obtenermos los datos de la fila cuyo indice es el 2
dfDesdeDict1.loc[2]
```

```
#Obtenemos los datos partir del nombre de la/s columna/s.
#Si son varias columnas se las pasa a traves de una lista
dfDesdeDict1['Notas']
```

```
dfDesdeDict1[['Nombre','Notas']]
```

Data Frames

- Métodos :
  - insert: nos premite agregar una nueva columna, siendo el dato que se inserta una lista.
  - Este método tiene tres parametros:
  - loc:le decimos donde agregar la columna
  - column: nombre de la nueva columna
  - value: la lista de datos a insertar

```
resultado = ['Desaprobado', 'Aprobado', 'Desaprobado', 'Desaprobado', 'Aprobado']
dfDesdeDict1.insert(loc = 3, column = 'Resultado', value = resultado)
print(dfDesdeDict1)
```

Data Frames

- Métodos :
  - **groupby**: Este metodo nos permite agrupar los datos por columnas. Es como el group by de SQL

```
dfDesdeDict3.groupby('Resultado').size()
```

```
dfDesdeDict3.groupby(['Apellido', 'Resultado'])['Id'].count()
```

Data Frames

- merge: Para poder unir dos data frames a traves de una o varias columnas se utiliza el método merge. Existen cuatro tipos de uniones:
    - inner: une solo lo que tienen en común ambos data frames pd.merge(df1, df2, how='inner', on=columnName)
    - left: unimos el df1 con el df2 sin perder información del df1 pd.merge(df1, df2, how='left', on=columnName)
    - right: unimos el df1 con el df2 sin perder información del df2 pd.merge(df1, df2, how='rigth', on=columnName)
    - outer: unimos ambos data frames sin perder nada de información pd.merge(df1, df2, how='outer', on=columnName)

donde columnName es el nombre o lista de nombres de columnas por los que queremos unir ambos data frames

```
dfTotal = dfDatosPersonales.merge(dfDesdeDict3, how = 'inner', on = ['Id'])
```

Data Frames

- Filtros:

```
dfTotal[dfTotal['Apellido'].isin(['Garcia', 'Gonzalez'])]
```

```
dfTotal[(dfTotal['Resultado'] == 'Desaprobado') & (dfTotal['Apellido'] == 'Garcia')]
    .sort_values('Notas',ascending=False)
```

Data Frames

- Métodos :
  - drop: Este metodo se utiliza eliminar o columnas del data frame. Este metodo tiene varios parametros, pero los mas importantes son: labels: indice o columna a borrar. Si son varios se los escribe dentro de una lista axis: toma dos valores:
    - 0 que se refiere a que la funcion drop se aplica a filas
    - 1 que se refiere a que la funcion drop se aplica a columnas

    por defecto el drop se aplica a las filas

```
dfDesdeDict2 = dfDesdeDict.drop(['Nombre', 'Apellido'], axis = 1)
```

```
#Borramos la primer fila cuyo indice es igual a 0
dfDesdeDict3 = dfDesdeDict.drop(0, axis = 0)
```

Data Frames

- unique: metodo que nos permite obtener los unicos elementos de una columna

```
print('Los datos unicos: ')
print(dfDesdeDict['Apellido'].unique().tolist())
```

Modulo Matplotlib

- Matplotlib es una de las bibliotecas más populares y completas de Python para la creación de gráficos y visualizaciones de datos en dos y tres dimensiones. Es ampliamente utilizada en ciencia, ingeniería, análisis de datos y educación gracias a su flexibilidad y potencia

- Tipos de gráficos soportados
  - Líneas (plot)
  - Dispersión (scatter)
  - Barras (bar, barh)
  - Pastel (pie)
  - Histogramas (hist)
  - Imágenes (imshow)
  - Superficies y gráficos 3D (plot_surface, scatter3D, etc.)

Modulo Seaborn

- Seaborn es una biblioteca de visualización de datos en Python diseñada para crear gráficos estadísticos atractivos y fáciles de interpretar. Está construida sobre Matplotlib y se integra de manera nativa con los DataFrames de Pandas, lo que facilita el análisis y la exploración de datos estructurados

- Tipos de gráficos soportados
  - Dispersión: scatterplot
  - Líneas: lineplot
  - Barras: barplot, countplot
  - Histogramas y densidades: histplot, kdeplot
  - Cajas y violines: boxplot, violinplot
  - Mapas de calor: heatmap
  - Gráficos de regresión: lmplot
  - Matriz de pares: pairplot
  - Gráficos conjuntos: jointplot

Unidad 5: Introducción al Machine Learning con Scikit-learn

TecnoFor by SNGULAR

Machine Learning

- Machine Learning (aprendizaje automático) es una rama de la inteligencia artificial (IA) que permite a las computadoras aprender de los datos y mejorar su rendimiento en tareas específicas sin ser programadas explícitamente para cada una de ellas

- El proceso básico de machine learning implica proporcionar a un algoritmo un conjunto de datos de entrada y, a menudo, los resultados esperados (salidas). El algoritmo analiza los datos, aprende las relaciones entre entradas y salidas, y ajusta sus parámetros internos para minimizar los errores en sus predicciones. Una vez entrenado, el modelo puede aplicar lo aprendido a nuevos datos para hacer predicciones o clasificaciones

Preparación de datos

- La preparación de datos implica transformar los datos brutos en un formato adecuado para el análisis y el modelado. Este proceso incluye varias tareas fundamentales:
  - **Limpieza de datos**: Eliminar o corregir errores, valores atípicos (outliers), duplicados y datos faltantes.
  - **Selección y transformación de variables**: Elegir las características relevantes y, si es necesario, crear nuevas variables a partir de las existentes.
  - **Codificación**: Convertir variables categóricas en valores numéricos (por ejemplo, mediante one-hot encoding).
  - **División de datos**: Separar el conjunto de datos en subconjuntos de entrenamiento, validación y prueba

Normalización de datos

- La normalización es una técnica de preprocesamiento que ajusta la escala de las variables numéricas para que sean comparables entre sí, evitando que las diferencias de magnitud entre variables afecten el aprendizaje del modelo.
- Por ejemplo, si una variable varía entre 0 y 1 y otra entre 10,000 y 100,000, la segunda podría dominar el proceso de modelado si no se normalizan ambas

Modulo Scikit-learn

- Scikit-learn (también conocida como sklearn) es una biblioteca gratuita y de código abierto para machine learning en Python, ampliamente utilizada tanto en la academia como en la industria. Fue desarrollada inicialmente por David Cournapeau en 2007 y lanzada públicamente en 2010, convirtiéndose en una de las herramientas más populares del ecosistema Python para aprendizaje automático

```python
from sklearn.ensemble import RandomForestClassifier

X = [[1, 2, 3], [11, 12, 13]]  # Datos de ejemplo
y = [0, 1]  # Etiquetas

clf = RandomForestClassifier(random_state=0)
clf.fit(X, y)
predicciones = clf.predict([[4, 5, 6]])
```

Unidad 6: Introducción a Numpy

TecnoFor by SNGULAR

Modulo Numpy

- NumPy es una de las librerías más importantes y utilizadas en Python para el cálculo numérico y la computación científica. Su principal característica es el manejo eficiente de arreglos (arrays) y matrices multidimensionales, permitiendo realizar operaciones matemáticas y estadísticas de forma rápida y sencilla, incluso con grandes volúmenes de datos

```
import numpy as np

a = np.array([1, 2, 3])              # Array unidimensional
b = np.array([[1, 2], [3, 4]])       # Array bidimensional (matriz)
c = np.zeros((3, 3))                 # Matriz 3x3 de ceros
d = np.ones((2, 2))                  # Matriz 2x2 de unos
e = np.arange(0, 10, 2)              # Array con valores de 0 a 8, paso 2
```

Tecnofor
by SNGULAR

¡Gracias!

Plaza de la Independencia, 8
28001-Madrid

www.tecnofor.es