diplomatura universitaria en
inteligencia artificial

FACULTAD DE CIENCIAS EXACTAS
UNIVERSIDAD NACIONAL DEL CENTRO DE LA PROVINCIA DE BUENOS AIRES

Introducción a Python

Conceptos básicos

Agenda

- Conceptos
- Elementos del lenguaje
- Estructuras
- Módulos y paquetes
- Funciones y clases

Elementos del lenguaje

- Lenguaje interpretado
- Alto nivel
- Multiplataforma
- Tipado dinámico
- Multiparadigma

Elementos del lenguaje

- Variables
  - `nombre_de_la_variable = valor_de_la_variable`
  - `print(mi_variable)`
- Tipos de datos
  - Número entero: `edad = 35`
  - Número entero octal: `edad = 043`
  - Número entero hexadecimal: `edad = 0x23`
  - Número real: `precio = 7435.28`
  - Booleano (verdadero / Falso):
    - `verdadero = True`
    - `falso = False`

Elementos del lenguaje

- Operadores aritméticos
  - + - * /
  - ** (exponente)
  - // (division entera) y % (modulo)
- Comentarios

```
>>> # Esto es un comentario de una sola  línea
>>> mi_variable = 15
>>> """Y este es un comentario  de varias
>>> líneas"""
```

Elementos del lenguaje

- Tipos de datos complejos
  - Tuplas: variable que permite almacenar varios datos inmutables.

```
>>> mi_tupla = ('cadena de texto', 15, 2.8, 'otro  dato', 25)
>>> print(mi_tupla[1]) # 15
>>> print(mi_tupla[1:4]) # (15, 2.8, 'otro dato')
>>> print(mi_tupla[3:]) # ('otro dato',  25)
>>> print(mi_tupla[:2]) # ('cadena de  texto', 15)
```

Elementos del lenguaje

- Listas

```
>>> mi_lista = ['cadena de texto', 15,  2.8, 'otro dato', 25]
>>> print(mi_lista[1]) # 15
>>> print(mi_lista[1:4]) # [15,  2.8, 'otro dato']
>>> print(mi_lista[-2]) # 'otro dato'
>>> mi_lista[2] = 3.8
>>> mi_lista.append('Nuevo Dato')
```

Elementos del lenguaje

- Diccionarios
  - `mi_mapa = {'clave_1': valor_1, 'clave_2': valor_2, 'clave_7': valor_7}`
  - `vacio = dict()`
  - `print(mi_mapa['clave_2']) # Salida: valor_2`
  - `del(mi_mapa ['clave_2'])`
  - `mi_mapa ['clave_1'] = 'Nuevo Valor'`

Elementos del lenguaje

- Identación
  - Obligatoria
  - 4 espacios en blanco
  - ```
    inicio de la estructura de control:
        expresiones
    ```
- Encoding
  - `# -*- coding: utf-8 -*-`
- Asignación múltiple
  - `a, b, c = 'string', 15, True`
    ```

Estructuras de control condicionales

- Operadores relacionales

  `==  !=  <  >  <=  >=`

- Operadores lógicos

  `and, or, xor`

```
>>> if semaforo == verde:
>>>     print("Cruzar la calle")
>>> else:
>>>     print("Esperar")
```

Estructuras de control iterativas

- while

```
>>> while True:
>>>     nombre = input("Indique su nombre: ")
>>>     if nombre:
>>>         break
```

Estructuras de control iterativas

- for

```
>>> mi_lista = ['Juan', 'Antonio', 'Pedro', 'Ivan']
>>> for nombre in mi_lista:
>>>     print(nombre)

>>> # -*- coding: utf-8 -*-
>>> for anio in range(2001, 2013):
>>>     print("Informes del Año " + str(anio))
```

Módulos y paquetes

- Módulos: archivos .py
- Paquete: agrupa módulos – Archivo de inicio __init__.py
- import
  - `import modulo # importar un módulo que no pertenece a un paquete`
  - `import paquete.modulo1 # importar un módulo que está dentro de un paquete`
- Alias
  - `import modulo as m`
  - `import paquete.modulo1 as pm`

Funciones

- Definición

```
>>> def  mi_funcion(nombre, apellido):
>>>    nombre_completo = nombre + ' ' + apellido
>>>    print(nombre_completo)
```

- Parámetros por defecto

```
>>> def saludar(nombre, mensaje='Hola '):
>>>    return mensaje + nombre

>>> saludar('Juan') # Imprime: Hola Juan
```

Funciones

- Retorno múltiple

```
>>> def mi_funcion():
>>>   return 'Hola', 12, [1,2,3]
>>> texto, numero, lista = mi_funcion()
```

Funciones

- Parámetros indeterminados por posición

```
>>> def indeterminados_posicion(*args):
...     for arg in args:
...         print arg
...
>>> indeterminados_posicion(5,"Hola",[1,2,3,4,5])
```

- Parámetros indeterminados por nombre

```
>>> def indeterminados_nombre(**kwargs):
...     for kwarg in kwargs:
...         print kwarg, "=>", kwargs[kwarg]
...
>>> indeterminados_nombre(n=5, c="Hola", l=[1,2,3,4,5])
```

Clases

```
>>> class Objeto(object):
>>>     atributo1 = 'a'
>>>     atributo2 = '1'

>>>     def _init_(self):
>>>         self.data = []

>>>     def metodo(self):
>>>         pass
```

Enlaces útiles

- http://docs.python.org.ar/tutorial/3/index.html
- https://docs.python.org/3/tutorial/

diplomatura universitaria en
inteligencia artificial

FACULTAD DE CIENCIAS
EXACTAS
UNIVERSIDAD NACIONAL DEL CENTRO
DE LA PROVINCIA DE BUENOS AIRES

Introducción a Python

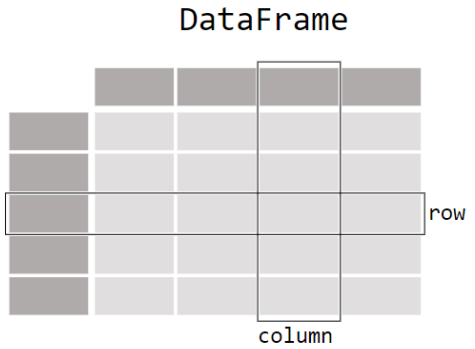
Librería Pandas

Agenda

- Conceptos
- Creación de DataFrames
- Funciones útiles
- Acceso a los datos
- Manipulación de los datos

Pandas

- API para el análisis y manejo de datos
  - Frameworks de ML soportan su uso como inputs.
  - Datos como tablas.
  - Estructuras de datos
    - DataFrame
    - Series
  - `import pandas as pd`



Creación de un DataFrame

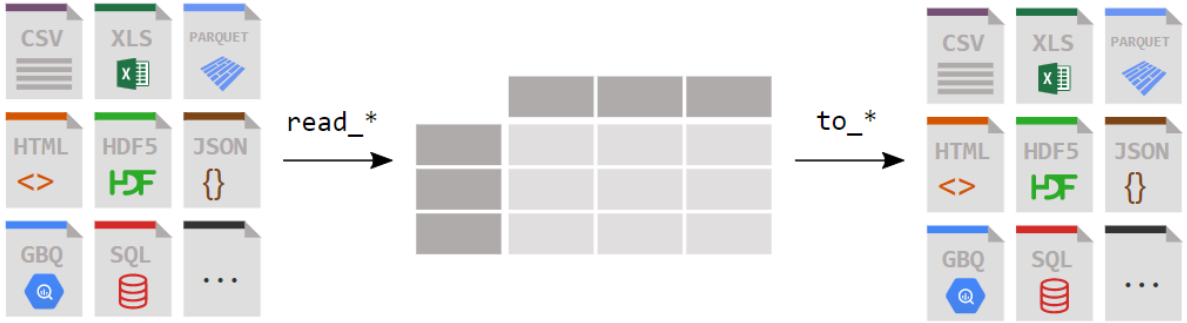
- Como un Diccionario mapeando nombres de columna a sus respectivas Series

```
>>> city_names = pd.Series(['San Francisco',
                            'San Jose',
                            'Sacramento'])
>>> population = pd.Series([852469, 1015785, 485199])

>>> pd.DataFrame({ 'City name': city_names,
                   'Population': population })
```

Creación de un DataFrame

- Desde un archivo
  - Tipos de datos soportados



Creación de un DataFrame

- Desde un archivo

```
>>> california_housing_dataframe = pd.read_csv("
        https://.../california_housing_train.csv", sep=",")
>>> california_housing_dataframe.describe()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000	17000.000000
mean	-119.562108	35.625225	28.589353	2643.664412	539.410824	1429.573941	501.221941	3.883578	207300.912353
std	2.005166	2.137340	12.586937	2179.947071	421.499452	1147.852959	384.520841	1.908157	115983.764387
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.790000	33.930000	18.000000	1462.000000	297.000000	790.000000	282.000000	2.566375	119400.000000
50%	-118.490000	34.250000	29.000000	2127.000000	434.000000	1167.000000	409.000000	3.544600	180400.000000
75%	-118.000000	37.720000	37.000000	3151.250000	648.250000	1721.000000	605.250000	4.767000	265000.000000
max	-114.310000	41.950000	52.000000	37937.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

Funciones útiles

- `head()` – muestra los primeros registros del DataFrame

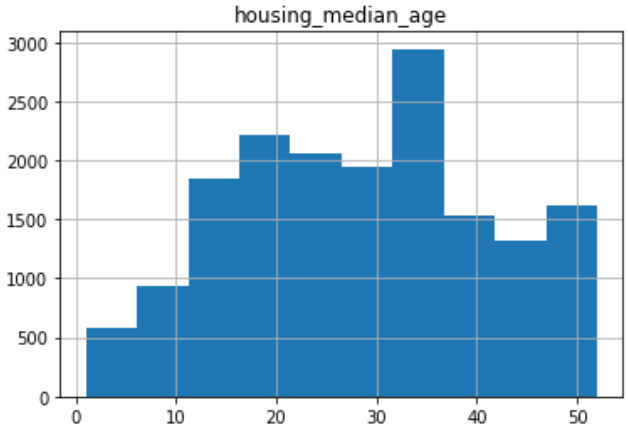
```
>>> california_housing_dataframe.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-114.31	34.19	15.0	5612.0	1283.0	1015.0	472.0	1.4936	66900.0
1	-114.47	34.40	19.0	7650.0	1901.0	1129.0	463.0	1.8200	80100.0
2	-114.56	33.69	17.0	720.0	174.0	333.0	117.0	1.6509	85700.0
3	-114.57	33.64	14.0	1501.0	337.0	515.0	226.0	3.1917	73400.0
4	-114.57	33.57	20.0	1454.0	326.0	624.0	262.0	1.9250	65500.0

Funciones útiles

- `hist(name)` – muestra la distribución de valores de una columna

```
>>> california_housing_dataframe.hist('housing_median_age')
```



Acceso a los datos

- Utilizando operaciones de diccionarios y listas

```
>>> cities = pd.DataFrame({ 'City name': city_names,
                            'Population': population })
>>> cities['City name']

0    San Francisco
1         San Jose
2       Sacramento
Name: City name, dtype: object
```

Acceso a los datos

- Utilizando operaciones de diccionarios y listas

```
>>> cities = pd.DataFrame({ 'City name': city_names,
                            'Population': population })
>>> cities['City name'][1]
'San Jose'

>>> cities[0:2]
```

	City name	Population
0	San Francisco	852469
1	San Jose	1015785

Manipulación de los datos

- Operaciones aritméticas sobre Series

```
>>> population / 1000

0     852.469
1    1015.785
2     485.199
```

Manipulación de los datos

- Funciones NumPy

```
>>> import numpy as np

>>> np.log(population)

0    13.655892
1    13.831172
2    13.092314
```

Manipulación de los datos

- Agregar Series a un DataFrame existente

```
>>> cities['Area square miles'] = pd.Series([46.87, 176.53, 97.92])

>>> cities['Population density'] = cities['Population'] /
                                   cities['Area square miles']
```

Manipulación de los datos

- Método `apply()` y funciones lambda

```
>>> population.apply(lambda val: val > 1000000)

0    False
1     True
2    False
```

Manipulación de los datos

- Método `apply()` y funciones lambda

```
>>> cities['Is wide and has saint name'] = (cities['Area square
        miles'] > 50) &
        cities['City name'].apply(lambda name:
                                  name.startswith('San'))
```

	City name	Population	Area square miles	Population density	Is wide and has saint name
0	San Francisco	852469	46.87	18187.945381	False
1	San Jose	1015785	176.53	5754.177760	True
2	Sacramento	485199	97.92	4955.055147	False

Enlaces útiles

- https://pandas.pydata.org/
- https://numpy.org/devdocs/user/quickstart.html

diplomatura universitaria en
inteligencia artificial

FACULTAD DE CIENCIAS
EXACTAS
UNIVERSIDAD NACIONAL DEL CENTRO
DE LA PROVINCIA DE BUENOS AIRES

Introducción a Python

Entorno Colab

Colaboratory

- Entorno gratuito de Jupyter Notebook
  - No requiere configuración.
  - Permite ejecutar en la nube, con acceso gratuito a GPUs.
  - Fácil de compartir.
  - https://colab.research.google.com/