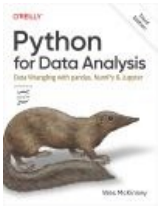
IMT 2200 - Introducción a Ciencia de Datos

Pontificia Universidad Católica de Chile
Instituto de Ingeniería Matemática y Computacional
Semestre 2023-1

Profesor: Rodrigo A. Carrasco

Clase 02: Clase 02: Repaso de Python y Jupyter Notebooks

Referencia: Capítulo 2 de "Python for Data Analysis", 3rd Ed., Wes McKinney y basado en el notebook introductorio de Eduardo Moreno



Python: Lenguaje de programación (ejecuta comandos, generalmente de un archivo)

Jupyter Notebooks: Interface Web para desplegar texto (Markdown) y comandos IPython (y otro lenguajes)

Expresiones y variables

In [ ]:
```
1+2
```

In [ ]:
```
a = 5
b = 3
a+b
```

In [ ]:
```
a+b
```

Información de funciones

In [ ]:
```
print?
```

Uso de librerías

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot([1,10,5,4])
```

Conceptos Básicos

Sintaxis

A diferencia de otros lenguajes el *scope* se define usando indentación y no llaves

In [ ]:
```
for x in [1,2,3]:
    if x < 2:
        print("Hola")
    else:
        print(x**2)
print("chao")
```

se usa el signo `#` para hacer comentarios

In [ ]:
```
for x in [1,2,3]:
    # si es menor que 2 dice hola
    if x < 2:
        print("Hola")
    # si no lo eleva al cuadrado
    else:
        print(x**2)
```

In [ ]:
```
print("Hola")  # comentario al final
```

Variables

```
In [ ]: a = 521
```

```
In [ ]: a = a + 1
```

```
In [ ]: a+ = 1
```

Todo es un *objeto* en Python, por lo que tienen atributos y métodos asociados.

```
In [ ]: a = "perro"
        a
```

```
In [ ]: a.upper()
```

```
In [ ]: a.count("r")
```

```
In [ ]: a.capitalize()
```

Es importantísimo recordar que con igualdad se crea una **referencia** entre objetos y no una copia.

```
In [ ]: a = [1, 2, 3]
        b = a
        b
```

```
In [ ]: a.append(4)
        a
```

```
In [ ]: b
```

```
In [ ]: b.append('H')
        b
```

```
In [ ]: a
```

```
In [ ]: b
```

Algunos objetos tienen un *metodo* que permite hacer copias del contenido.

```
In [ ]: a = [1, 2, 3]
        b=a.copy()
        a.append(4)
        a
```

```
In [ ]: b
```

Tipos y referencias fuertes

Python no tiene "tipos" fuertes asociado a una variable. Una variable puede cambiar de un tipo a otro, a veces sin siquiera darnos cuenta.

```
In [ ]: a = 5
        type(a)
```

```
In [ ]: b = 'foo'
        type(b)
```

```
In [ ]: '5' + 5
```

```
In [ ]: a+a
```

```
In [ ]: '5'+'5'
```

```
In [ ]: "hola" + "chao"
```

```
In [ ]: b
```

```
In [ ]: b*10
```

Algunas conversiones las hace internamente Python:

```
In [ ]: a = 4.5
        type(a)
```

```
b = 2
type(b)
a/b
type(a/b)
```

In [ ]:
```
v = [1,2,3]
v*5
```

Atributos y métodos

```
In [1]: a = 'foo'

In [2]: a.<Tab>
a.capitalize  a.format      a.isupper     a.rindex      a.strip
a.center      a.index       a.join        a.rjust       a.swapcase
a.count       a.isalnum     a.ljust       a.rpartition  a.title
a.decode      a.isalpha     a.lower       a.rsplit      a.translate
a.encode      a.isdigit     a.lstrip      a.rstrip      a.upper
a.endswith    a.islower     a.partition   a.split       a.zfill
a.expandtabs  a.isspace     a.replace     a.splitlines
a.find        a.istitle     a.rfind       a.startswith
```

In [ ]:
```
a = '36'
```

In [ ]:
```
a.
```

In [ ]:
```
a.replace('3','H')
```

Operaciones binarias:

Operacion	Descripción
a + b	Suma
a - b	Resta
a * b	Multiplicacion
a / b	División
a // b	División entera
a % b	Módulo o resto de la división entera
a ** b	Potencia
a & b	Y lógico
a | b	O lógico
a ^ b	O-exclusivo lógico
a == b	True si son iguales
a != b	True si no son iguales
a <= b	True si a es menor o igual que b (también con <=, <, >)
a is b	Trues si a y b son una referencia al mismo objeto
a is not b	Trues si a y b no son una referencia al mismo objeto

In [ ]:
```
False ^ False
```

In [ ]:
```
a = True
type(a)
```

In [ ]:
```
5 - 7
12 + 21.5
5 <= 2 & 7>3
```

In [ ]:
```
2 in a
```

In [ ]:
```
a = [1, 2, 3]
b = a
c = a.copy()
c
```

In [ ]:
```
a is c
```

In [ ]:
```
a == c
```

In [ ]:
```
a = 2
a is None
```

Tipos (Types) básicos de Python

Tipos (Types) básicos de Python

Los tipos mas relevantes son:

- `None` : nulo, para identificar algo que no existe
- `str` : String, texto
- `bytes` : datos en formato ASCII
- `bool` : booleano, puede ser verdadero (True) o falso (False)
- `int` : número entero
- `float` : número "real" (mas bien, doble-precision 64 bits)

Números

In [ ]:
```
ival = 17239871
ival ** 6
```

In [ ]:
```
fval = 7.243
fval2 = 6.78e-5
fval2
```

In [ ]:
```
3 / 2
```

In [ ]:
```
7 // 3
```

In [ ]:
```
7 % 3
```

Strings (Texto)

Puede usarse comillas (") o apostrofe ('). Si son múltiples lineas, usar triple comillas o triple apostrofe

In [ ]:
```
a = 'one way of writing a string'
b = "another way"
b
```

In [ ]:
```
c = """
This is a longer string that
spans multiple lines
"""
```

In [ ]:
```
c
```

In [ ]:
```
print(c)
```

Se pude convertir de un tipo a otro (usualmente, un número en un string)

In [ ]:
```
a = 3
print("El valor es" + a)
```

In [ ]:
```
a = 3
str(a)
print("El valor es " + str(a))
```

In [ ]:
```
a='6.7'
float(a)+1
```

In [ ]:
```
a = 3
print("El valor es %d" % a)
```

Booleanos

Verdadero o Falso

In [ ]:
```
True & True
False or True
```

Type casting

Las funciones `float()` , `int()` , `bool()` y `str()` nos permiten cambiar entre estos tipos

In [ ]:
```
s = '3.74159'
fval = float(s)
type(fval)
int(fval)
bool(fval)
```

None

Tipo especial para representar algo que no existe

In [ ]:
```
a = None
a is None
b = 5
b is not None
```

Una gracia de Python es que una operación se interpreta de acuerdo al tipo de la variable.

In [ ]:
```
5 + 7
```

In [ ]:
```
'5' + '7'
```

In [ ]:
```
'5' * 5
```

Otros tipos

Fechas y horas

A través de las librerias de Python, podemos acceder a nuevos tipos, que incluyen sus propias funciones y métodos. Por ejemplo, para datos es muy util para manejar fechas. En este caso, importando la libreria `datetime`

In [ ]:
```
from datetime import datetime, date, time
dt = datetime(2021, 4, 12, 9, 34, 0)
dt.date()
```

In [ ]:
```
dt.date()
dt.time()
```

In [ ]:
```

```

In [ ]:
```
dt.strftime('%d/%m/%Y %H:%M')
```

In [ ]:
```
dt
```

In [ ]:
```
datetime.strptime('2009-10-31', '%Y-%m-%d')
```

In [ ]:
```
dt.replace(minute=0, second=10)
```

In [ ]:
```
dt2 = datetime(2021, 4, 12, 9, 36)
delta = dt2 - dt
delta
```

In [ ]:
```
delta.total_seconds()
```

Control de Flujo

if, elif, y else

Ejecuta un comando si (`if`) se cumple una condición, o si (`elif`) otra condicion se cumple, y si ninguna se cumple (`else`) ejecutar otro comando

In [ ]:
```
x = 0
if x < 0:
    print('Negativo')
else:
    print('No negativo')
```

In [ ]:
```
x = 6.2
if x < 0:
    print('Negativo')
elif x == 0:
    print('igual a cero')
elif 0 < x < 5:
    print('positivo menor que 5')
else:
    print('positivo mayor o igual que 5')
```

ciclos for

Ejecuta una serie de comando para (`for`) un grupo de elementos de un conjunto.

In [ ]:
```
for i in [1,2,3,4]:
    print(i)
```

In [ ]:
```
for i in [4,"Perro",2,1]:
    print(i)
```

In [ ]:
```
for i in [4, [2,3] , "Perro"]:
    print(i+i)
```

Puedo interrumpir un ciclo usando `break`

In [ ]:
```
for i in [1,2,3,4]:
    for j in [1,2,3,4]:
        if i==j:
            break
        print((i, j))
```

comando util: `range` : Algo muy util para ciclos de `for` es el comando `range()` , que entrega una secuencia de números

In [ ]:
```
list(range(8))
```

In [ ]:
```
range?
```

In [ ]:
```
print(list(range(4,10,2)))
```

In [ ]:
```
range(10)
list(range(10))
```

In [ ]:
```
for i in range(5):
    print (i)
```

In [ ]:
```
list(range(0, 20, 2))
```

In [ ]:
```
list(range(50, 0, -7))
```

Expresiones Ternarias (Ternary expression)

Python permite poner expresiones como `if` o `for` dentro de una linea, o una definición de una variable.

In [ ]:
```
x = -2
if x > 0:
    print("Hola")
else:
    print("Chao")
```

In [ ]:
```
x = -2
print("Hola"  if x > 0  else "Chao")
```

para hacer listas (estructura que veremos pronto) mas facilmente

In [ ]:
```
[i**2 for i in range(5) ]
```

Extra: texto en idioma español (u otros idiomas)

Trabajo con acentos y eñes

El texto, por defecto, es un código llamado ASCII. Este código no considera caracteres especiales como ñ, acentos, u otras letras. Estas tipicamente se "codifican" como string, siendo el mas común el UTF8. Sin embargo, hay varios encoding distintos (por ejemplo, *UTF16*, o *Latin1*). A veces, al leer datos, pueden estar en la codificación incorrecta, pero podemos codificarlos (`encode`) o decodificarlos (`decode`) como queramos.

In [ ]:
```
a = "baño"
a
```

In [ ]:
```
a8 = a.encode('utf8')
a8
```

In [ ]:
```
a16 = a.encode('utf16')
a16
```

In [ ]:
```
a32 = a.encode('latin1')
a32
```

In [ ]:
```
print(a8)
```

In [ ]:
```
print(a8.decode('utf8'))
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js