Explicación Práctica Python 2023

- Jupyter Notebook
  - ¿Qué es?
  - Instalación
  - Formas de utilización
  - Algunas funcionalidades
  - Exportar
- Manejo de string y armado de estructuras
- Buenas prácticas
  - nombres de variables, funciones
  - recorrido con for
- match case más que un reemplazo de if/else
- F-string nos ayuda a...

Jupyter Notebook

En la guía subida al portal encontrarán algunas consideraciones generales para tener en cuenta.

¿Qué es?

- Es una herramienta flexible y dinámica que permite la ejecución de código simple a modo de guía y visualización.
- permite ejecutar código python y otros lenguajes.
- cada celda tiene la posibilidad de configurarse para código ejecutable o markdown.
- la ejecución es lineal y las variables se guardan a lo largo de todo el notebook.
- pueden encontrar repositorios de proyectos en github que ponen notebook con ejemplos para ejecutar y probar.
- la utilización de widgets permite agregar funcionalidades al notebook

Instalación

- Puede instalarse en forma local
- Puede instalarse como servidor y generar cuentas para usuaries
- Jupyter-Lab es una versión con más funcionalidades que aún está desarrollándose

Formas de utilizarlo

Local

- al instalarlo poder ejecutarlo desde una terminal

```
$jupyter notebook
```

- se inicia en un navegador web
- podes crear, modificar archivos, carpetas, exportar desde donde ejecutas la orden

Remoto

- utilizando un servidor instalado y con cuenta creada.
- google-colab es una opción de acceso online y compartido, soporta gran cantidad de librerías además de python, hay que verificar las versiones que se usan.
- conectar el repositorio de código público con binder

Algunas funcionalidades

- Atajos:
  - Ejecutar una celda y pasara la siguiente → Shift + Enter
  - Ejecutar una celda y permanecer en la misma → Ctrl + Enter
  - Ejecutar una celda y crear una nueva -> Alt + Enter
  - Salir del modo edición para pasar al modo selección → Escape
  - Salir del modo selección para entrar al modo edición → Enter
  - Agregar una celda debajo de la actual → modo selección + ”b”
  - Agregar una celda arriba de la actual → modo selección + ”a”
  - Convertir una celda de código a markdown → modo selección + ”m”

In [ ]:
```
?
```

- Instalando librerías

In [ ]:
```
pip install pandas
```

- Usando código de lenguajes en celdas markdown:

```
$pip install pandas
print(f'hola')
```

- Comandos especiales - magic

In [ ]:
```
%timeit L = [n ** 2 for n in range (1000) ]
```

In [ ]:
```
%%timeit
L = []
for n in range (1000) :
    L.append (n ** 2)
```

Comparación eficiencia entre map y for

In [ ]:
```
%%timeit
numeros = [1,2,3,4,5,6,7,8,9]
numeros_dupl = []
for cada in numeros:
    numeros_dupl.append(cada*2)
numeros_dupl
```

In [ ]:
```
numeros = [1,2,3,4,5,6,7,8,9]
%timeit numeros_map_dupl = map(lambda x:x*2, numeros)
```

In [ ]:
```
def dupl(x):
    return x*2
%timeit numeros_map_dupl = map(dupl, numeros)
```

In [ ]:
```
%timeit L = [n * 2 for n in numeros ]
```

- El formato de las celdas de markdown está definido por las características del mismo:
  - Agregar imágenes
  - listas
  - links
  - videos

- No es necesario realizar un print para ver el contenido de una variable

Manejo de string y armado de estructuras

In [ ]:

```
texto = '''NumPy is the fundamental package needed for scientific computing with Pytho

    Website: https://www.numpy.org
    Documentation: https://numpy.org/doc
    Mailing list: https://mail.python.org/mailman/listinfo/numpy-discussion
    Source code: https://github.com/numpy/numpy
    Contributing: https://www.numpy.org/devdocs/dev/index.html
    Bug reports: https://github.com/numpy/numpy/issues
    Report a security vulnerability: https://tidelift.com/docs/security

It provides:

    a powerful N-dimensional array object
    sophisticated (broadcasting) functions
    tools for integrating C/C++ and Fortran code
    useful linear algebra, Fourier transform, and random number capabilities

Testing:

    NumPy versions ≥ 1.15 require pytest
    NumPy versions < 1.15 require nose

Tests can then be run after installation with:

python -c 'import numpy; numpy.test()'

Call for Contributions

The NumPy project welcomes your expertise and enthusiasm!

Small improvements or fixes are always appreciated; issues labeled as "good first issu

Writing code isn’t the only way to contribute to NumPy. You can also:

    review pull requests
    triage issues
    develop tutorials, presentations, and other educational materials
    maintain and improve our website
    develop graphic design for our brand assets and promotional materials
    translate website content
    help with outreach and onboard new contributors
    write grant proposals and help with other fundraising efforts

If you’re unsure where to start or how your skills fit in, reach out! You can ask on t

Our preferred channels of communication are all public, but if you’d like to speak to

We also have a biweekly community call, details of which are announced on the mailing

If you are new to contributing to open source, this guide helps explain why, what, and

Powered by NumFOCUS
About

The fundamental package for scientific computing with Python.
numpy.org
Topics
python numpy
Resources
Readme
License
BSD-3-Clause License
Releases 183
v1.20.1 Latest
on 7 Feb
```

```
+ 182 releases
Sponsor this project

    @numfocus
    numfocus NumFOCUS

    tidelift tidelift.com/funding/github/pypi/numpy

    https://numpy.org/about/

Learn more about GitHub Sponsors
Packages
No packages published
Used by 617k

    @sktatsuno
    @Andy-CH-BO-AN
    @sinablk
    @bartmch
    @dydwkd486
    @mochibbb
    @ishujais
    @khoa-luan

+ 617,044
Contributors 1,083

    @charris
    @teoliphant
    @mattip
    @cournape
    @eric-wieser
    @seberg
    @pearu
    @rgommers
    @pv
    @juliantaylor
    @mwiebe

+ 1,072 contributors
Languages

Python 63.4%
C 35.3%
C++ 1.0%
JavaScript 0.1%
Fortran 0.1%
Shell 0.1%'''
texto.split(' ')
```

In [ ]:
```
texto_mod = texto.splitlines()
texto_mod
```

- Encontrar las líneas que contengan un caracter específico:

In [ ]:
```
for i in texto_mod:
    if '@' in i.strip().lower():
        print(i)
```

Si en lugar de encontrar los que contienen **sólo queremos los colaboradores del proyecto, los que comienzan con @**

In [ ]:
```
for i in texto_mod:
    if  i.strip().startswith('@'):
        print(i.strip())
```

Armado de estructuras

Dado un string que contiene datos de los puntajes de un juego, realice un programa que permita obtener los puntajes ordenados según el nivel que se desee

In [ ]:
```
puntajes_nivel ='''maria: 160, uno
Ailen: 45, uno
Ana: 154, dos
Dolores: 186, tres
Inti: 185, dos
Iona: 181, dos
Jimena: 162, tres
Laura: 86, uno
Ludmila: 23, uno
Luján: 117, dos
Celina: 47, dos
Mariana: 174, tres
Mercedes: 152 tres'''
```

¿Cómo separamos el string?

In [ ]:
```
pn = puntajes_nivel.split('\n')
```

In [ ]:
```
pn
```

- Generamos una estructura que permita acceder más facilmente

In [ ]:
```
datos ={}
for linea in pn:
    nom, punt, nivel = linea.split()
    print(nom.split(':')[0])
    print(punt.split(',')[0].strip())
    print(nivel.strip())
```

In [ ]:
```
datos ={}
for linea in pn:
    nom, punt, nivel = linea.split()
    datos[nom.split(':')[0]] = {'puntaje': int(punt.split(',')[0].strip()), 'nivel':ni
datos
```

- ¿Cómo podemos ordenar nuestros datos?

Ordenar las keys

In [ ]:
```
sorted(datos)
```

- Ordenar por puntaje

In [ ]:
```
datos.keys()
```

In [ ]:
```
datos.values()
```

In [ ]:
```
datos.items()
```

In [ ]:
```
sorted(datos.items(), key= lambda x:x[1]['puntaje'])
```

- Ordenar por nivel

In [ ]:
```
sorted(datos.items(), key= lambda x:x[1]['nivel'])
```

- Obtener los del nivel "uno"

In [ ]:
```
nivel_uno = list(filter(lambda x:x[1]['nivel'] =='uno', datos.items()))
```

In [ ]:
```
nivel_uno
```

In [ ]:
```
def por_nivel(dat, n):
    return list(filter(lambda x:x[1]['nivel'] ==n, datos.items()))

nivel_ingre = input('ingrese el nivel: (uno, dos,tres)')
while not nivel_ingre=='':
    print(por_nivel(datos, nivel_ingre))
    nivel_ingre = input('ingrese el nivel: (uno, dos,tres)')
```

Y con zip?

In [ ]:
```
mediciones_enero = [3,4,5,6]
mediciones_septiembre = [13,43,62,73]
localidades = ['Paraná', 'Gualeguay', 'Victora', 'Villa Elisa']
```

In [ ]:
```
mediciones_ene_sep = zip(mediciones_enero, mediciones_septiembre)
```

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

In [ ]:
```
print(list(mediciones_ene_sep))
```

Si lo ejecutamos nuevamente

In [ ]:
```
print(tuple(mediciones_ene_sep))
```

¿Qué pasó?

zip genera un iterador por lo tanto una vez que lo accedemos el iterador apunta al final, para no perderlo debemos guardarlo en una estructura

In [ ]:
```
localidades_mediciones = list(zip(localidades,mediciones_enero, mediciones_septiembre
```

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

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

Y si quisiéramos generar una estructura tipo diccionario?

In [ ]:
```
localidades_mediciones_dicc= {i:[j, k] for i, j, k in zip(localidades,mediciones_enero
```

In [ ]:
```
localidades_mediciones_dicc
```

También podríamos tener distintos tipos de estructuras, tuplas y listas

In [ ]:
```
mediciones_enero = (3,4,5,6)
mediciones_septiembre = [13,43,62,73]
localidades = ['Paraná', 'Gualeguay', 'Victora', 'Villa Elisa']
```

In [ ]:
```
localidades_mediciones_dicc= {i:[j, k] for i, j, k in zip(localidades,mediciones_enero
```

In [ ]:
```
localidades_mediciones_dicc
```

Buenas prácticas

- ¿Cómo definimos los nombres de las variables, funciones, son claras al momento de leer el código?

In [ ]:
```
lista = []
for i in range(3):
    #guardo el nombre
    x = input("Ingrese el nombre:")

    #guardo el apellido
    y = input("Ingrese el apellido: ")

    #guardo la edad
    z = input("ingrese la edad: ")
    lista.append((x,y,z))

#recorro para imprimir
for i in range(len(lista)):
    print(lista[i])
```

- Cambiar nombres de variables: nom, apelliedo, edad, personas(en lugar de lista)

In [ ]:
```
#buena forma de recorrer para imprimir
for elem in lista:
    print(elem)
```

In [134…
```
#salvo que sea necesario no es bueno recorrer asi tampoco:
cant = len(lista)
for i in range(cant):
    print(lista[i])
```

In [ ]:
```
def otra_funcion3():
    #funcion para el ingreso de datos
    for i in range(3):
    #guardo el nombre
        x = input("Ingrese el nombre:")

    #guardo el apellido
        y = input("Ingrese el apellido: ")

        #guardo la edad
        z = input("ingrese la edad: ")
        lista.append((x,y,z))
    return lista
lista_1 = funcion_1()
```

- Cambiar nombre de función y variables como habíamos visto antes

Match case más pattern matching que if/else

In [ ]:
```
numero = int(input("Ingrese un número: "))
match numero:
    case 1:
        print("Uno")
    case 2:
        print("Dos")
    case _:
        print("Otra cosa")
```

In [ ]:
```
def avanzar(distancia):
    print(f"Robot avanza {distancia} cm ")


def girar(grados):
    print(f"Robot gira {grados} grados")


def frenar():
    print("Robot frena")
```

In [ ]:
```
while True:
    orden = input("Comando: ").split()
    match orden:
        case ["avanzar", distancia]:
            avanzar(int(distancia))
        case ["girar", grados]:
            girar(float(grados))
        case ["frenar"]:
            frenar()
        case ["salir"]:
            break
        case otracosa:
            print(f"No entiendo el comando: {otracosa}")
```

In [ ]:
```
patrones= [True, False, [], None, 0]
for i in patrones:
    if i == True:
        print(f'encontre True')
    else:
        print(f'encontre False')
```

In [ ]:
```
patrones= [True, False, [], None, 0]
for i in patrones:
    match i:
        case True:
            print(f'encontre True')
        case False:
            print(f'encontre False')
        case _:
            print(f'otra cosa')
```

In [ ]:
```
list(range(2,20,4))
```

Ejemplos de Daniel Moisset, video contando un poco más.

Si quisiera encontrar los números entre 1 y 100 que son múltiplos de 3 y 5, solo de 3, solo de 5 o de ninguno, ¿cómo haríamos?

- Primero casos más generales
- Luego los más específicos

In [ ]:
```
for i in range(1, 101):
    if (i % 3 == 0 ) and (i % 5 == 0):
        print("ehh sii")
    elif (i % 3 == 0 ):
         print("ehh")
    elif (i % 5 == 0):
         print("sii")
    else:
        print(i)
```

In [ ]:
```
for i in range(1, 101):
    match (i % 3 == 0 ), (i % 5 == 0):
        case True, True: print("ehh sii")
        case True, _: print("ehh")
        case _, True: print("sii")
        case _: print(i)
```

In [ ]:
```
points = [(0, 0), (0,5), (5,0), (5,5), ('f',8), ('f')]
for point in points:
    match point:
        case (0, 0):
            print("Origin")
        case (0, y):
            print(f"Y={y}")
        case (x, 0):
            print(f"X={x}")
        case (x, y):
            print(f"X={x}, Y={y}")
        case _:
            raise ValueError("Not a point")
```

F-string y ...

- Datetime
- Debugging
- En list comprehension

- Datetime

In [ ]:
```
import datetime
hoy = datetime.datetime.today()
```

In [ ]:
```
hoy
```

In [ ]:
```
print(f"{hoy:%d-%m-%Y}")
```

In [ ]:
```
print(f"{hoy:%Y}")
```

In [ ]:
```
print(f"{hoy:%F}")
```

- Debugging

In [ ]:
```
x  = 10
y = 25
print(f"x = {x} , y = {y}")
```

Podemos hacerlo más simple?

In [ ]:
```
print(f"{x = }, {y = }")
```

Y si queremos agregarle decimales

In [ ]:
```
print(f"{x = :.3f}, {y = }")
```

In [ ]:
```
valor = 1234567890

print(f"{valor:,}")
```

In [ ]:
```
area = 1973.9208802178716
print(f'El area es: {area:,.2f}')
```

Usando el módulo locale

In [ ]:
```
import locale

locale.setlocale(locale.LC_ALL, '')
print(locale.format_string('%.2f', area, grouping=True, monetary=True))
```

In [ ]:
```
print(f'El area es: {area:.2f}')
```

- En list comprehension

In [ ]:
```
personas = ['Alicia', 'Juana', 'Lucía', 'Paula', 'Carolina']
saludos = [f'Hola {p}!' for p in personas]
saludos
```

In [ ]:
```
nombres = ['Ana', 'Juan', 'Lucía', 'Carlos', 'María', 'Julia', 'Marta']
patron = 'an'
nombres_con_patron = [f'{nombre} contiene el patrón' for nombre in nombres if patron :
print(nombres_con_patron)
```