Annotated follow-along guide_Hello, Python!

January 25, 2024

1 Annotated follow-along guide: Hello, Python!

This notebook contains the code used in the instructional videos from Week 1: Hello, Python!.

1.1 Introduction

This follow-along guide is an annotated Jupyter Notebook organized to match the content from each week. It contains the same code shown in the videos for the week. In addition to content that is identical to what is covered in the videos, you'll find additional information throughout the guide to explain the purpose of each concept covered, why the code is written in a certain way, and tips for running the code.

As you watch each of the following videos, an in-video message will appear to advise you that the video you are viewing contains coding instruction and examples. The in-video message will direct you to the relevant section in the notebook for the specific video you are viewing. Follow along in the notebook as the instructor discusses the code.

To skip directly to the code for a particular video, use the following links:

1. **Section ??**
2. **Section ??**
3. **Section ??**
4. **Section ??**
5. **Section ??**
6. **Section ??**

1. Discover more about Python

[9]:
```
# Print to the console.
print("Hello, world!")
```

```
Hello, world!
```

[10]:
```
# Print to the console.
print(22)
```

```
22
```

[11]:
```
# Simple arithmetic
(5 + 4) / 3
```

[11]: 3.0

[12]:
```
# Assign variables.
country = 'Brazil'
age = 30

print(country)
print(age)
```

```
Brazil
30
```

[13]:
```
# Evaluations
# Double equals signs are used to check equivalency.
10**3 == 1000
```

[13]: True

[15]:
```
# Evaluations
# A single equals sign is reserved for assignment statements.
10 ** 3 = 1000
```

```
  File "<ipython-input-15-8baa8abf97f4>", line 3
    10 ** 3 = 1000
               ^
SyntaxError: can't assign to operator
```

[7]:
```
# Evaluations
# Double equals signs are used to check equivalency.
10 * 3 == 40
```

[7]: False

[8]:
```
# Evaluations
# Double equals signs are used to check equivalency.
10 * 3 == age
```

[8]: True

[9]:
```
# Conditional statements
if age >= 18:
```

```
    print('adult')
else:
    print('minor')
```

```
adult
```

[10]:
```
# Loops
for number in [1, 2, 3, 4, 5]:
    print(number)
```

```
1
2
3
4
5
```

[11]:
```
# Loops
my_list = [3, 6, 9]

for x in my_list:
    print(x / 3)
```

```
1.0
2.0
3.0
```

[12]:
```
# Functions
def is_adult(age):

    if age >= 18:
        print('adult')
    else:
        print('minor')
```

[13]:
```
# Use the function that was just created.
is_adult(14)
```

```
minor
```

[14]:
```
# Use the built-in sorted() function.
new_list = [20, 25, 10, 5]

sorted(new_list)
```

[14]:
```
[5, 10, 20, 25]
```

2. Jupyter Notebooks

NOTE: The import statements cell must be run before running some of the following cells. This setup step was not shown in the instructional video, but you will learn about import statements later in this course.

[37]:
```
# Import statements.
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
```

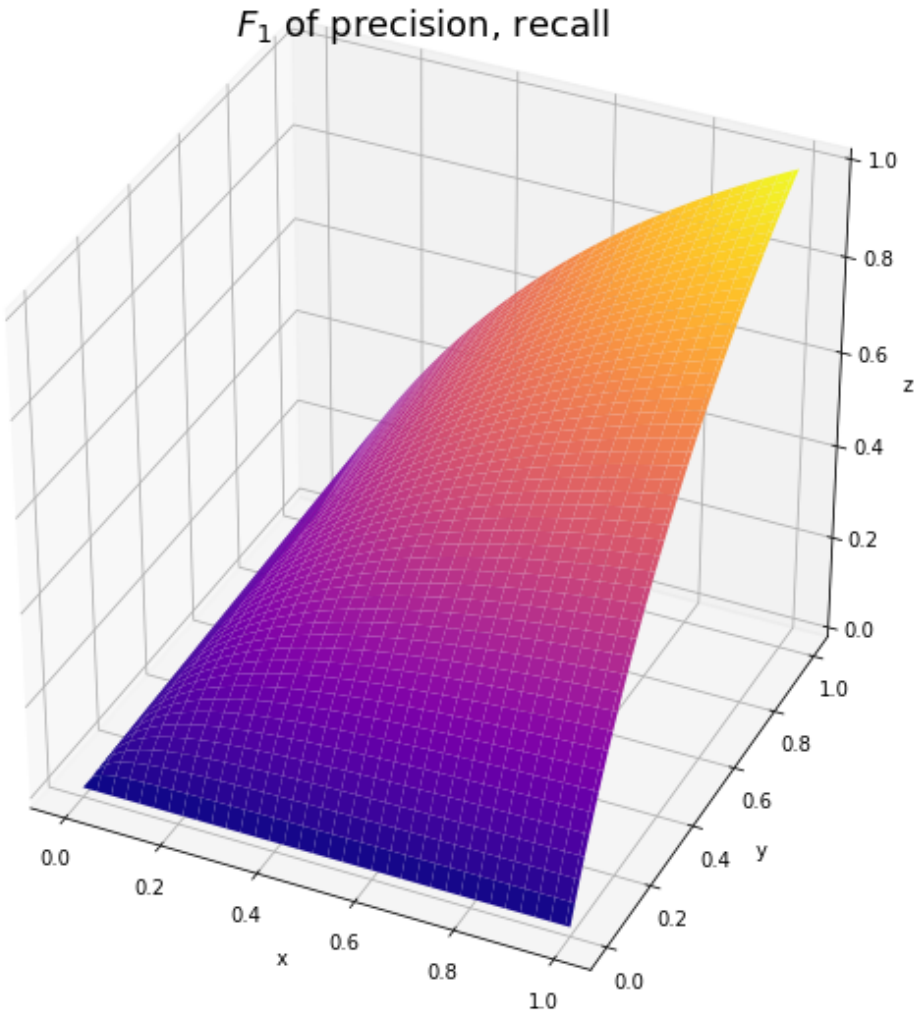
[38]:
```
# Create a list.
my_list = [10, 'gold', 'dollars']
```

[39]:
```
# Use the helper function to calculate F1 score used in the following graphics.
def f1_score(precision, recall):
    score = 2*precision*recall / (precision + recall)
    score = np.nan_to_num(score)

    return score
```

[40]:
```
# Generate a graph of F1 score for different precision and recall scores.
x = np.linspace(0, 1, 101)
y = np.linspace(0, 1, 101)
X, Y = np.meshgrid(x, y)
Z = f1_score(X, Y)
fig = plt.figure()
fig.set_size_inches(10, 10)
ax = plt.axes(projection='3d')
ax.plot_surface(X, Y, Z, rstride=2, cstride=3, cmap='plasma')

ax.set_title('$F_{1}$ of precision, recall', size=18)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.view_init(35, -65)
```



NOTE: The following cells use markdown (like this cell) to create formatted text like headers and bullets, tables, and mathematical equations. You can select any cell and enter into edit mode to view the markdown text. Then run the cell to view the rendered output.

1.1.1 Section 2

- Part 1:
- Part 2:

Title	Author	Date
The Art of War	Sun Tzu	5th cent. BCE
Don Quixote de la Mancha	Miguel de Cervantes Saavedra	1605
Pride and Prejudice	Jane Austen	1813

$$\int_0^\infty \frac{x^3}{e^x - 1}\, dx = \frac{\pi^4}{15}$$

3. Object-oriented programming

[41]:
```
# Assign a string to a variable and check its type.
magic = 'HOCUS POCUS'
print(type(magic))
```

```
<class 'str'>
```

[42]:
```
# Use the swapcase() string method to convert from caps to lowercase.
magic = 'HOCUS POCUS'
magic = magic.swapcase()
magic
```

[42]:
```
'hocus pocus'
```

[43]:
```
# Use the replace() string method to replace some letters with other letters.
magic = magic.replace('cus', 'key')
magic
```

[43]:
```
'hokey pokey'
```

[ ]:
```

```

[44]:
```
# Use the split() string method to split the string into two strings.
magic = magic.split()
magic
```

[44]:
```
['hokey', 'pokey']
```

[45]:
```
# Set up the cell to create the `planets` dataframe.
# (This cell was not shown in the instructional video.)
import pandas as pd
data = [['Mercury', 2440, 0], ['Venus', 6052, 0,], ['Earth', 6371, 1],
        ['Mars', 3390, 2], ['Jupiter', 69911, 80], ['Saturn', 58232, 83],
        ['Uranus', 25362, 27], ['Neptune', 24622, 14]
]

cols = ['Planet', 'radius_km', 'moons']

planets = pd.DataFrame(data, columns=cols)
```

[46]:
```
# Display the `planets` dataframe.
planets
```

[46]:

	Planet	radius_km	moons
0	Mercury	2440	0
1	Venus	6052	0
2	Earth	6371	1
3	Mars	3390	2
4	Jupiter	69911	80
5	Saturn	58232	83
6	Uranus	25362	27
7	Neptune	24622	14

[47]:
```
# Use the shape dataframe attribute to check the number of rows and columns.
planets.shape
```

[47]: (8, 3)

[48]:
```
# Use the columns dataframe attribute to check column names.
planets.columns
```

[48]: Index(['Planet', 'radius_km', 'moons'], dtype='object')

4. Variables and data types

[5]:
```
# Assign a list containing players' ages.
age_list = [34, 25, 23, 19, 29]
```

[6]:
```
# Find the maximum age and assign to `max_age` variable.
max_age = max(age_list)
max_age
```

[6]: 34

[17]:
```
# Convert `max_age` to a string.
max_age = str(max_age)
max_age
```

[17]: '34'

[18]:
```
# Reassign the value of `max_age`.
max_age = 'ninety-nine'
max_age
```

[18]: 'ninety-nine'

[8]:
```
# FIRST, RE-RUN THE SECOND CELL IN THIS VIDEO.
# Check the value contained in `max_age` (SHOULD OUTPUT 34).
max_age
```

[8]: 34

[7]:
```
# Find the minimum age and assign to `min_age` variable.
min_age = min(age_list)

# Subtract `min_age` from `max_age`
max_age - min_age
```

[7]:
```
15
```

5. Create precise variable names

[55]:
```
# Trying to assign a value to a reserved keyword will return a syntax error.
else = 'everyone loves some esparagus'
```

```
  File "<ipython-input-55-1f1f078fc2a2>", line 2
    else = 'everyone loves some esparagus'
         ^
SyntaxError: invalid syntax
```

[56]:
```
# The word "asparagus" is misspelled. That's allowed.
esparagus = 'everyone loves some esparagus'
```

[57]:
```
# Order of operations
2 * (3 + 4)
```

[57]:
```
14
```

[58]:
```
# Order of operations
(2 * 3) + 4
```

[58]:
```
10
```

[59]:
```
# Order of operations
3 + 4 * 10
```

[59]:
```
43
```

6. Data types and conversions

[60]:
```
# Addition of 2 ints
print(7+8)
```

```
15
```

[61]:
```
# Addition of 2 strings
print("hello " + "world")
```

```
hello world
```

[62]:
```
# You cannot add a string to an integer.
print(7+"8")
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

<ipython-input-62-199724c0b4c0> in <module>
  1 # You cannot add a string to an integer
----> 2 print(7+"8")


TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

2 The type() function checks the data type of an object.

type("A")

[64]:
```
# The type() function checks the data type of an object.
type(2)
```

[64]:
```
int
```

[65]:
```
# The type() function checks the data type of an object.
type(2.5)
```

[65]:
```
float
```

[66]:
```
# Implicit conversion
print(1 + 2.5)
```

```
3.5
```

[67]:
```
# Explicit conversion (The str() function converts a number to a string.)
print("2 + 2 = " + str(2 + 2))
```

```
2 + 2 = 4
```

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.