PythonBasics

October 7, 2024

1 Getting Familiar with Notebooks

[1]:
```
print("Hello SU")
# this code print Hello SU
```

```
Hello SU
```

1.0.1 the code above prints "HELLO SU"

2 Working with numbers

[ ]:
```
x=30
y=15
z=x+y
print(z)
```

I: Change values and re-execute

[ ]:
```
x
```

[ ]:
```
print(x)  #contrasting raw output vs print output - not too much difference for␣
↪numbers
```

3 Strings and Substrings

[1]:
```
a="hello"
a=a+" there" #using + as an overloaded operator that will concatenate strings
a
```

[1]:
```
'hello there'
```

I:Contrast raw output vs print output

[ ]:
```
print(a)
```

I: Stop here and save the code written so far as a python file. Download the file and execute it. Contrast the "friendliness" of JupyterHub vs. the simple execution of the python code

[2]:
```
s='Syracuse'
```

[3]:
```
s[3] #REMEMBER - Numbering in python starts from 0 !!!
```

[3]:
```
'a'
```

[ ]:
```
s[0]
```

[4]:
```
s[1:4] #substring, starts in 1, stop at 4 but does not include element in␣
↪position 4
```

[4]:
```
'yra'
```

[ ]:
```
s[0:8]
```

[ ]:
```
s[2:] #substring starting from position 2 until the end of the string - it␣
↪includes item in position 2
```

[6]:
```
s[:4] #substring from the beginning up to position 4 not including item in␣
↪position 4
```

[6]:
```
'Syra'
```

[5]:
```
s[-2] #get the second character from the end
```

[5]:
```
's'
```

[ ]:
```
len(s)
```

4 Formating

[1]:
```
"One, %d, three" %2
```

[1]:
```
'One, 2, three'
```

[8]:
```
result=1.2345
```

[9]:
```
"One, %d, three" %result  #The %d option is only for integers
```

[9]:
```
'One, 1, three'
```

[10]:
```
"One, %f, three" %result #The %f option is for floating point numbers
```

[10]:
```
'One, 1.234500, three'
```

[11]:
```
"One, %.3f, three" %result
```

[11]: 'One, 1.234, three'

4.0.1 F-Strings

In Python 3.7, f-strings were introduced to make it easier to format string literals in the `print()` statement.

Here's how it works:

- Put an `f` in front of the string literal, like this: `f"`
- For any variable you want to print, enclose in `{curly braces}` within the string literal.
- At run-time the variable in `{curly braces}` is replaced with its value! This is called **string interpolation**.

For example:

[ ]:
```
name = "Mary"
major = "Data Science"
gpa = "4.0"
print(f"{name} is a {major} major. Her gpa is {gpa}")
```

4.1 Formatting with F-Strings

The other method of formatting data in Python is F-strings. As we saw in the last lab, F-strings use interpolation to specify the variables we would like to print in-line with the print string.

You can format an f-string

- `{var:d}` formats `var` as integer
- `{var:f}` formats `var` as float
- `{var:.3f}` formats `var` as float to 3 decimal places.

Example:

[2]:
```
name ="Ann"
wage = 10
print(f"{name} makes ${wage:.2f} per hour")
```

```
Ann makes $10.00 per hour
```

5 (Side note) Jupyter Hub shortcuts

Press the ESC (escape key) and then the letter H

I:Switch to slides

6 Part 2

Variable names must start with letters and are case sensitive

Variables that start with _ or ___ (double underscore) are python specific

Python keywords cannot be used as variable names

```
[ ]: var1=10  #nothing seems to happen but an assignment has been made
```

```
[ ]: var1  #checking on the value stored in var1
```

```
[ ]: x, y, z = 1, 2, 3    #multiple variable assignment
```

```
[ ]: y
```

7 Strings and substrings 2

NOTE - IMPORTANT: In python, indexed elements start in position 0

python string methods: https://docs.python.org/2.5/lib/string-methods.html a friendlier view https://www.w3schools.com/python/python_ref_string.asp

```
[ ]: a="cisco switch"
```

```
[ ]: a.index("i")  #find location for the first instance of the letter i
```

```
[ ]: a.count("i") #count number of instances of the letter i in object/string a
```

```
[ ]: a.find("sco") #find the index where the pattern starts
```

```
[ ]: a.find("xyz")
```

```
[ ]: a.startswith("c") #boolean result true or false
```

strip() removes starting and ending whitespaces in a string by default. You can also specify a specific character happening at the start of end of the string to be removed

Q. The strip() method seems useful for what key aspect of a data analysis pipeline?
A. *Data cleaning*

```
[ ]: b = " Cisco Switch "
```

```
[ ]: b.strip()
```

```
[ ]: c = "$$$$Cisco Switch$$$$"
```

```
[ ]: c.strip("$") #strip '$' characters from the beginning and end of the string
```

Replacing string elements

```
[ ]: b
```

```
[ ]: b.replace(" ","")
```

[ ]:
```
b
```

[ ]:
```
k=b.replace(" ","")
k
```

Splitting a string character .. it returns a list

[ ]:
```
d="cisco,hp,juniper"
```

[ ]:
```
d.split(",")
```

Joining strings

[ ]:
```
x="Cisco"
```

[ ]:
```
y="2691"
```

[ ]:
```
x+y
```

Testing for the presence of a character

[ ]:
```
"o" in x
```

[ ]:
```
"b" in x
```

8 Working with lists

Lists can have any data type and they are indexed. List elements are mutable (they can be changed)

[ ]:
```
list1=["Cisco","Avaya",10,10.5,-11]
```

[ ]:
```
len(list1)
```

[ ]:
```
list1[2]
```

[ ]:
```
list1.append(100) #add the value 100 to the end of the list
```

[ ]:
```
list1
```

[ ]:
```
del list1[4]
```

[ ]:
```
list1
```

[ ]:
```
list1.insert(2,"HP")
```

[ ]:
```
list1
```

```
[ ]: list1.pop(0)
```

```
[ ]: list1
```

```
[ ]: list1.remove(10)
```

```
[ ]: list1
```

```
[ ]: list2=[1,2,3,"a","b","c"]
```

```
[ ]: list2
```

```
[ ]: list2[0:3]  #an example of slicing - this will be very important and we will␣
     ↪expand on it later
```

```
[ ]: list2[:3]
```

```
[ ]: list2[::2]  #every 2nd element after the first
```

Sets are an ordered collection of unique elements (there are no repetitions)

```
[ ]: set1=set([11,12,13,14,15,15,15,11])
```

```
[ ]: set1
```

```
[ ]: type(set1)
```

```
[ ]: type(b)
```

```
[ ]: set2={14,15}
```

```
[ ]: set2
```

```
[ ]: set2.add(16)
```

```
[ ]: set2
```

```
[ ]: set1.intersection(set2) #common elements between sets
```

```
[ ]: set1.union(set2)
```

9 Working with tuples

```
[ ]: my_tuple=(1,2,3,4,5)
```

```
[ ]: my_tuple
```

[ ]:
```
my_tuple[1]
```

[ ]:
```
my_tuple[1]=5
```

[ ]:
```
3 in my_tuple  #Checking for elements in a tuple
```

[ ]:
```
7 in my_tuple
```

10 Working with dictionaries

An unordered set of key value pairs Dictionaries are indexed by the key. Each key should be unique and immutable.

[12]:
```
d1={"Vendor": "Cisco", "Model": "2600", "IOS": "12.4", "Ports":"4"}
```

[13]:
```
d1
```

[13]:
```
{'Vendor': 'Cisco', 'Model': '2600', 'IOS': '12.4', 'Ports': '4'}
```

[14]:
```
d1["IOS"]
```

[14]:
```
'12.4'
```

[15]:
```
d1["Vendor"]
```

[15]:
```
'Cisco'
```

[16]:
```
del d1["Ports"]
```

[17]:
```
d1
```

[17]:
```
{'Vendor': 'Cisco', 'Model': '2600', 'IOS': '12.4'}
```

[18]:
```
"IOS" in d1
```

[18]:
```
True
```

[19]:
```
d1.keys()
```

[19]:
```
dict_keys(['Vendor', 'Model', 'IOS'])
```

[20]:
```
d1.values()
```

[20]:
```
dict_values(['Cisco', '2600', '12.4'])
```

[21]:
```
d1.items()
```

[21]: `dict_items([('Vendor', 'Cisco'), ('Model', '2600'), ('IOS', '12.4')])`

10.1 Basic interactions with Jupyter Hub

1. Save the notebook and download it to your local PC
2. Change the name of the notebook (Add a 2 to the name or something like that) and upload it
3. Delete the recently uploaded notebook