Python Fundamentals day 21

Today's Agenda

- String replication
- String formatting-Conversion
- 'f' string literal
- Raw string literal
- Regular expressions

String Replication

We know what is string concatenation, let us know what is string replication with the help of following example where we are trying to replicate the string python three times

```
s="PYTHON"
s_rep=s*3 # * is the replication operator in python
print(s)
print(s_rep)
```

Output:

```
In [19]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
PYTHON
PYTHONPYTHONPYTHON
```

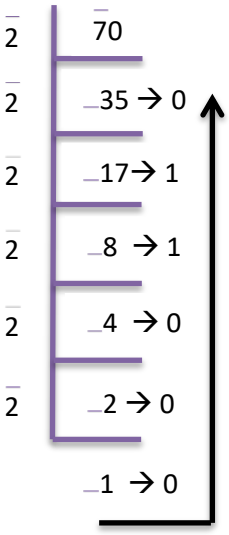
We can repeat the single string value the amount of times equivalent to the integer value passed.

String formatting- Conversion

- **b- decimal to binary**

```
a=70
print(a)
print("{0:b}".format(a)) #converting decimal to binary
```

Logic:



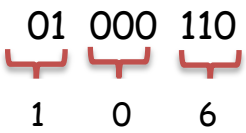
Output:

```
In [20]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
70
1000110
```

- o- decimal to octal

```
a=70
print(a)
print("{0:o}".format(a)) #converting decimal to octal
```

Logic:



Output:

```
In [21]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
70
106
```

- x- decimal to hexadecimal

```
a=70
print(a)
print("{0:x}".format(a)) #converting decimal to hexadecimal
```

Logic:

0100 0110

4 6

Output:

```
In [22]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
70
46
```

'f' string literal

As of Python 3.7, f-strings are a great new way to format strings. Not only are they more readable, more concise, and less prone to error than other ways of formatting, but they are also faster!

Let us understand by an example

format()

```
import math
name='rohit'
place='Blore'
print("{} {}".format(name,place))
print("{1} {0}".format(name,place))
print("{0:.4f}".format(math.pi))
```

Output:

```
In [25]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
rohit Blore
Blore rohit
3.1416
```

f-string

```
import math
name='rohit'
place='Blore'
print(f"{name} {place}")
print(f"{place} {name}")
print(f"{math.pi:.4f}")
```

```
In [27]:

In [27]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
rohit Blore
Blore rohit
3.1416
```

Let us compare the performance of both as below

```
import timeit
print(timeit.timeit(stmt="{0:.2f}".format(3.1416), number=1000))
print(timeit.timeit(stmt=f"{3.1416:.2f}", number=1000))
```

Output:

```
In [35]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
9.599985787644982e-06
3.100000321865082e-05
```

As we can see f-string is much better than format() from above output. The timing changes with respect to the processor.

Raw string literal

Let us see an example to understand the meaning of raw string

```
name="Ro\nhit" #normal string
name=r"Ro\nhit" #raw string
print(name)
```

Output:

```
In [36]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
Ro
hit

In [37]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
Ro\nhit
```

Regular Expressions

Regular expression is a sequence of characters that forms a search pattern. Why do we need it? Where it is used? Let us see

- In **MS word** whenever there is a spelling mistake, we get red underline which notifies us that the spelling is wrong. This is done by matching the word we enter to the actual word and if there is a mismatch then it notifies by the red line.

- **Google search engine**, where anything that we entered is matched with whatever is there in google. It does this with the help of software called as webscroller who scans the entire internet to find the search queries matching the entered data.
- **Spam filtering** is another best example where all our mails are being monitored and filtered based on the words present in the content of it. Every mail goes through the spam filter where if words like **free, discount, cheap, offer etc** are encountered then it is a spam.

Let us see an example and understand in a better way

```
import re

text="Python is easy"
regex=r"Python" # regular exp should always be a raw string,
match=re.match(regex,text)
print(match)
```

Output:

```
In [39]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
<re.Match object; span=(0, 6), match='Python'>
```

The output says that match is found in the span of 0 to 6 that is Python. Let us see how to access that in a better way

```
import re

text="Python is easy"
regex=r"Python" # regular exp should always be a raw string,
match=re.match(regex,text)
start,end=match.span()
print(text[match.start():match.end():])
```

Output:

```
In [41]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
Python
```

Let us now try to find another match

```
import re

text="Python is super easy"
regex=r"super" # regular exp should always be a raw string.
match=re.match(regex,text)
```

Output:

```
In [45]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
None
```

We can see that we did not get any match object. Which means the match() starts searching from the beginning and if it doesn't find the match in starting then it doesn't return any match object.

Is there a way to overcome this disadvantage? Definitely there is, let us see how

```
import re

text="Python is super easy"
regex=r"super" # regular exp should always be a raw string
print(re.search(regex,text))
```

Output:

```
In [46]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
<re.Match object; span=(10, 15), match='super'>
```

By making use of search(), irrespective of the position the matching is done. So we have found a match object in span of 10 to 15 where 15 is exclusive.

```
import re

text="Python is super easy"
regex=r"super" # regular exp should always be a raw string
match=re.search(regex,text)
print(text[match.start():match.end():])
```

Output:

```
In [47]: runfile('C:/Users/rooman/OneDrive/
Desktop/python/test.py', wdir='C:/Users/rooman/
OneDrive/Desktop/python')
super
```

This is how we slice the match object.

From the above examples we understood that match() can be used for pattern matching where the match object is present in the beginning whereas search() will do the same irrespective of the position of the match object.