By Sai Kumar

Formatting the Strings

- We can format the strings with variable values by using replacement operator **{}** and **format()** method.

- The main objective of **format()** method to format string into meaningful Output form.

Case- 1: Basic formatting for default, positional and keyword arguments

```
name = 'Sai'

salary = 100000

age = 24

print("{} 's salary is {} and his age is
{}".format(name,salary,age))

print("{0} 's salary is {1} and his age is
{2}".format(name,salary,age))

print("{x} 's salary is {y} and his age is
{z}".format(z=age,y=salary,x=name))
```

Case-2: Formatting Numbers

d → Decimal Integer

f → Fixed point number(float).The default precision is 6

b → Binary format

o → Octal Format

x → Hexa Decimal Format (Lower case)

X → Hexa Decimal Format (Upper case)

Eg-1:

```
print("The integer number is: {}".format(123))
print("The integer number is: {:d}".format(123))
print("The integer number is: {:5d}".format(123))
print("The integer number is: {:05d}".format(123))
```

Eg-2:

```
print("The float number is: {}".format(123.4567))
print("The float number is: {:f}".format(123.4567))
print("The float number is: {:8.3f}".format(123.4567))
print("The float number is: {:08.3f}".format(123.4567))
print("The float number is: {:08.3f}".format(123.45))
print("The float number is: {:08.3f}".format(786786123.45))
```

Note:

- {:08.3f}
- Total positions should be minimum 8.
- After decimal point exactly 3 digits are allowed. If it is less then 0s will be placed in the last positions
- If total number is < 8 positions then 0 will be placed in MSBs
- If total number is >8 positions then all integral digits will be considered.
- The extra digits we can take only 0

Note: For numbers default alignment is Right Alignment (>)

Eg-3: Print Decimal value in binary, octal and hexadecimal form

```
print("Binary Form:{0:b}".format(153))
print("Octal Form:{0:o}".format(153))
print("Hexa decimal Form:{0:x}".format(154))
print("Hexa decimal Form:{0:X}".format(154))
```

Note: We can represent only int values in binary, octal and hexadecimal and it is not possible for float values.

Note:

1) {:5d} It takes an integer argument and assigns a minimum width of 5.

2) {:8.3f} It takes a float argument and assigns a minimum width of 8 including "." and after decimal point exactly 3 digits are allowed with round operation if required

3) {:05d} The blank places can be filled with 0. In this place only 0 allowed.

Case-3: Number formatting for signed numbers

- While displaying positive numbers, if we want to include + then we have to write {:+d} and {:+f}
- Using plus for -ve numbers there is no use and for -ve numbers - sign will come automatically.

```
print("int value with sign:{:+d}".format(123))
print("int value with sign:{:+d}".format(-123))
print("float value with sign:{:+f}".format(123.456))
print("float value with sign:{:+f}".format(-123.456))
```

Case-4: Number formatting with alignment

- <, >, ^ and = are used for alignment
- < → Left Alignment to the remaining space
- ^ → Canter alignment to the remaining space
- > → Right alignment to the remaining space
- = → Forces the signed (+) (-) to the left most position

Note: Default Alignment for numbers is Right Alignment.

Ex:

```
print("{:5d}".format(12))
print("{:<5d}".format(12))
print("{:<05d}".format(12))
print("{:>5d}".format(12))
print("{:>05d}".format(12))
print("{:^5d}".format(12))
print("{:=5d}".format(-12))
print("{:^10.3f}".format(12.23456))
print("{:=8.3f}".format(-12.23456))
```

Case-5: String formatting with format()

Similar to numbers, we can format String values also with format() method.

s.format(string)
print("{:5d}".format(12))
print("{:5}".format("rat"))
print("{:>5}".format("rat"))
print("{:<5}".format("rat"))
print("{:^5}".format("rat"))
print("{:*^5}".format("rat")) #Instead of * we can use any character(like +,$,a etc)

Note: For numbers default alignment is right where as for strings default alignment is left

Case-6: Truncating Strings with format() method

print("{:.3}".format("saikpsoftware"))
print("{:5.3}".format("saikpsoftware"))
print("{:>5.3}".format("saikpsoftware"))
print("{:^5.3}".format("saikpsoftware"))
print("{:*^5.3}".format("saikpsoftware"))

Case-7: Formatting dictionary members using format()

person={'age':24,'name':'Sai'}

print("{p[name]}'s age is: {p[age]}".format(p=person))

Note: p is alias name of dictionary

person dictionary we are passing as keyword argument

More convenient way is to use **person

person={'age':24,'name':'Sai'}
print("{name}'s age is: {age}".format(person))**

Case-8: Formatting class members using format()

class Person:
age=48
name="Sai"
print("{p.name}'s age is :{p.age}".format(p=Person()))

By Sai Kumar

```
class Person:
def __init__(self,name,age):
self.name=name
self.age=age
print("{p.name}'s age is :{p.age}".format(p=Person('Sai',24)))
print("{p.name}'s age is :{p.age}".format(p=Person('Ravi',50)))
```

Note: Here Person object is passed as keyword argument. We can access by using its reference variable in the template string

Case-9: Dynamic Formatting using format()

```
string="{:{fill}{align}{width}}"
print(string.format('cat',fill='*',align='^',width=5))
print(string.format('cat',fill='*',align='^',width=6))
print(string.format('cat',fill='*',align='<',width=6))
print(string.format('cat',fill='*',align='>',width=6))
```

Case-10: Dynamic Float format template

```
num="{:{align}{width}.{precision}f}"
print(num.format(123.236,align='<',width=8,precision=2))
print(num.format(123.236,align='>',width=8,precision=2))
```

Case-11: Formatting Date values

```
import datetime
#datetime formatting
date=datetime.datetime.now()
print("It's now:{:%d/%m/%Y %H:%M:%S}".format(date))
```

Case-12: Formatting complex numbers

complexNumber=1+2j

```
print("Real Part:{0.real} and Imaginary
Part:{0.imag}".format(complexNumber))
```

Output: Real Part: 1.0 and Imaginary Part: 2.0