MTA 98-381
Lesson 9
Print, Random & Others

Number Print Formatting

```
# integer with min 4 width.
print("'{:4d}'".format(23))          # '  23'

# Pad with 0.
print("'{:04d}'".format(23))         # '0023'

# 2 decimal point float.
print ('{:.2f}'.format (45.6))       # 45.60
print ('%.2f' % (45.6))              # 45.60

# Thousand separator.
print ('{:,.2f}'.format (12345.6)) # 12,345.60
```

String Print Formatting

```
# left aligned.
print("'{:4}'".format('ab'))        # 'ab  '
print("'{:<4} '".format('ab'))      # 'ab  '
print("%-4s" % ('ab'))              # 'ab  '

# right aligned
print("'{:>4}'".format('ab'))       # '  ab'
print("%4s" % ('ab'))               # '  ab'

# center aligned, pad with '_'
print("'{:_^4}'".format('ab'))      # '_ab_'
```

DateTime Print Formatting

```
from datetime import datetime
now = datetime.now()
print ('{:%Y-%B-%d %H:%M:%S %p %A}'.format(now))
# 2019-May-01 13:06:36 PM Saturday

print ('{:%y-%b-%d %I:%M:%S %p %a}'.format(now))
# 19-May-01 01:06:36 PM Sat

# American style date, May 01, 19
print ('{%B %d, %y}'.format(now))
```

Random Integers

- Must import random module.
- `random.randrange()`
  - generates a random integer in `range()`:

```
import random
random.randrange (4)                    # 0,1,2,3
random.randrange (2, 4)                 # 2,3
random.randrange (1, 10, 2)             # 1,3,5,7,9
```

  - start <= n < stop with step

```
n = random.randrange (start=0, stop, step=1)
```

Random Integers (cont.)

- `random.randint (a, b)`
  - Same as `random.randrange (a, b+1)`

```
import random
random.randint (2, 4)                    # 2,3,4
random.randrange (2, 4)                  # 2,3
```

Random Real Numbers

- 0.0 <= n <= 1.0

```
n = random.random()
```

- start <= n <= stop

```
n = random.uniform (start, stop)
```

Exercise: Random Numbers

- What is the Python code to generate the following random numbers?
- Q1. 0, 1, 2, 3, 4, 5
- Q2. 3, 4, 5, 6
- Q3. 0, 3, 6, 9
- Q4. 3, 5, 7, 9, 11
- Q5. Float between 2 and 5?

eval() Function

- Allows us to execute a string as Python code. It accepts a source string and returns an object.
- Examples:

```
a = eval ("5")                  # a is int 5.
b = eval ("2 == 2")             # b is bool True.
c = input ("Enter an expression:")  # 2**3
print (c)                       # 2**3
print (eval(c))                 # 8
```

math Module

- Allows us to access common math functions and constant.
- Examples:

```
import math
print (5/4)                # 1.25
print (round(5/4))         # 1
print (math.ceil(5/4))     # 2
print (math.floor(5/4))    # 1
# print (int(5/4))         # 1. Another way to floor.
# print (5//4)             # 1. Another way to floor.
print (math.sqrt(9))       # 3.0
print (math.pow(2,4))      # 16.0
# print (2**4)             # 16. Another way to power.
print (math.pi)            # 3.141592653589793
```

Stripping Space in String

- *string*.`strip()` – strip whitespace from both the left and right sides of the string.
- *string*.`lstrip()` – strip whitespace from the left side of the string.
- *string*.`rstrip()` – strip whitespace from the right side of the string.
- *string*.`strip('c')` – strip character 'c' from both the left and right sides of the string.
- *string*.`rstrip(' c')` – strip space and 'c' from both the right side of the string.

Exercise: Stripping from String

- What is the output?

```
s1 = "   ** MMU FCI **   "
print ("0.'{}'".format(s1))
print ("1.'{}'".format(s1.lstrip()))
print ("2.'{}'".format(s1.rstrip()))
print ("3.'{}'".format(s1.strip()))
print ("4.'{}'".format(s1.strip('*')))
print ("5.'{}'".format(s1.strip(' *')))
print ("6.'{}'".format(s1.strip('* ')))
print ("7.'{}'".format(s1.strip('* M')))
```

Command Line Arguments

- If we run a Python code file named cmdargs.py from Command Prompt, the command is:

  ```
  py cmdargs.py
  ```

  - or

  ```
  python cmdargs.py
  ```

  - Environment variable PATH must be correctly set in order for the above command to work.
- Command line arguments are the string(s) we enter after the Python command.

Command Line Arguments (cont.)

- Example (3 arguments):

```
py cmdargs.py 1 world
```

  - `cmdargs.py`, `1` and `world` are the arguments.
- To obtain the command line arguments in Python code, use `sys.argv`.
- `sys.argv` return a list of command line arguments.
- Example:

```
import sys
print ('Number of arguments:', len(sys.argv))
print ('Argument List:', sys.argv)
```

Exercise: Command Line Arguments

- Write a Python code that sum the command line arguments.
- Sample input:

```
py sumargs.py 4 6 8 10
```

  - Output: `Sum = 28`

`pass` Statement

- The `pass` statement is a null operation (it does nothing).
- It is used as a placeholder for future code.
- It provides a way to fulfil some syntax requirement temporarily.
- For examples, writing the following codes would generate syntax errors:

```
def myfunc():

if 2 < 3:
else:

while 2 < 3:
```

`pass` Statement *cont.*

- We can use `pass` statement to get rid of the syntax errors temporary:

```
def myfunc():
  pass

if 2 < 3:
  pass
else:
  pass

while 2 < 3:
  pass
```