modules (math)

January 25, 2023

1 Modules in Python

- There are several modules in Python.
- These modules can be used for specific purposes depending upon requirement of user
- One should only know what are the functions present in the module and how to use them effectively to work with modules
- Here are some modules and their purpose
  - math -> to work with some mathematical functions
  - csv -> to work with comma sepearated value kind of data
  - fpdf -> to work with PDFs
  - pandas, numpy -> for data science
  - django, jinja2 -> web development
  - matplotlib, seaborn, ggplot -> for plotting
  - keras, tersorflow, scikit-learn, pytorch, opencv -> machine learning, deep learning, AI, image processing
- In order to use a module contents we must import that module first.

```
import module_name
```

1.1 math.factorial()

[ ]:
```
import math # some functions and constant values
print(math.factorial(5))
```

[2]:
```
factorial(5) # not a buil
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [2], in <cell line: 1>()
----> 1 factorial(5)

NameError: name 'factorial' is not defined
```

1.2 math.gcd()

[3]:
```
import math
print(math.gcd(12, 18)) # # all these functions in modules
# are version dependent
# New in version 3.5
```

```
6
```

[8]:
```
import math
print(math.gcd(12, 18, 21, 2))
# 6 21
# 3 2 --> 1
```

```
1
```

1.3 math.lcm()

[9]:
```
import math
print(math.lcm(2, 3))
```

```
6
```

[10]:
```
import math
print(math.lcm(2, 3, 4))
```

```
12
```

[17]:
```
import math
lst = list(range(1, 21))
print(math.lcm(*lst))
# 2 3 --> 6
# 6 4 --> 12
# 12 5 --> 60
```

```
232792560
```

1.3.1 Unpacking operator

- * symbol is used as unpacking operator

[20]:
```
lst = [10, 20, 30] # packing
print(*lst) # 10, 20, 30
```

```
10 20 30
```

[22]:
```
lst = [10, 20, 30] # packing
print(*lst, sep = '\n')
```

```
10
20
30
```

[25]:
```
import math
lst = [1, 2, 3]
math.lcm(*lst) # unpacking operator
```

[25]: 6

1.4 sqrt()

[29]:
```
import math
print(math.sqrt(64)) # will give floating point value
```

```
8
```

[30]:
```
import math
print(math.isqrt(64))
```

```
8
```

1.5 math.ceil()

- Rounds up the given floating point value to nearest integer

[31]:
```
import math
x = 12.2
print(math.ceil(x))
```

```
13
```

[33]:
```
import math
print(math.ceil(126.000000000001))
```

```
127
```

[37]:
```
import math
print(math.ceil(14.0))
```

```
14
```

1.6 math.floor()

- Rounds down the given floating point value to nearest integer

[32]:
```
import math
x = 12.2
print(math.floor(x))
```

```
12
```

[34]:
```
import math
print(math.floor(19.9999999999))
```

```
19
```

1.7 math.radians()

- Converts given degrees into radians

[39]:
```
import math
print(math.radians(90))
```

```
1.5707963267948966
```

1.8 math.sin()

[43]:
```
import math
print(math.sin(math.radians(30))) # sin 90 degrees but in radians
print(math.tan(math.radians(45)))
```

```
0.49999999999999994
0.9999999999999999
```

1.9 math.perm()

- Returns in how many ways we can arrange k entities from n entities with order

[47]:
```
import math
print(math.perm(4, 3)) # with order
# npr --> n! / (n-r)!
# ncr --> n! / (n-r)! * r!
```

```
24
```

[50]:
```
import math
math.factorial(4) / math.factorial(4-3)
```

[50]: 24.0

1.10 math.comb()

- In how many ways we can arrange k entities from n entities order doesn't matter

[53]:
```
import math
print(math.comb(3, 2))
```

```
3
```

[54]:
```
import math
math.factorial(3) / (math.factorial(3-2) * math.factorial(2))
```

[54]: 3.0