ADVANCED PYTHON PROGRAMMING

Modules and Packages

Peter Lo

Python Modules Overview

- There are three different ways to define a module in Python:
  - A module can be written in Python itself.
  - A module can be written in C and loaded dynamically at run-time, like the re (regular expression) module.
  - A built-in module is intrinsically contained in the interpreter, like the math module.

What is Module?

- A module is a file containing Python definitions and statements.
- The file name is the module name with the suffix .py appended.
- Within a module, the module's name (as a string) is available as the value of the global variable __name__.

The import Statement

- Module contents are made available to the caller with the import statement.

```
Python

import <module_name>
```

- An alternate form of the import statement allows individual objects from the module to be imported directly into the caller's symbol table:

```
Python

from <module_name> import <name(s)>
```

The import Statement (cont.)

- It is also possible to import individual objects but enter them into the local symbol table with alternate names:

```
Python

from <module_name> import <name> as <alt_name>[, <name> as <alt_name> …]
```

Built-in Standard Modules

- There are plenty of built in modules, just as there are with built in functions.
- This is where Python really excels! It takes what's called the "batteries included" approach.
- The standard Python Module can be found from:
  - https://docs.python.org/3/py-modindex.html

Module platform

- The platform module is used to retrieve as much possible information about the platform on which the program is being currently executed.
- This module plays a crucial role when you want to check whether your program is compatible with the python version installed on a particular system or whether the hardware specifications meet the requirements of your program.

```
# List the system detail
platform.uname()
```

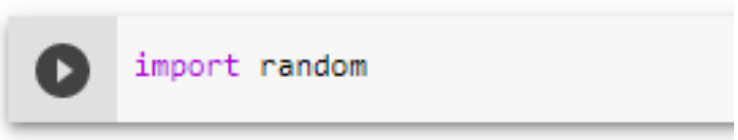
```
uname_result(system='Linux', node='cfe857dd37bc', release='4.19.112+', version='#1 SMP Thu Jul 23 08:00:38 PDT 2020', machine='x86_64', processor='x86_64')
```

```
# Display the Python version
platform.python_version()
```

```
'3.6.9'
```

Module random

- This module implements pseudo-random number generators for various distributions.



- Detail functions information are listed in:
  - https://docs.python.org/3/library/random.html

Function randint

- The ***randint()*** method returns an integer number selected element from the specified range (including both start number and end number).
  - ***randint(*** *Start*, *End* ***)***

```
# Generate a random number between 1-10 (both included)
random.randint(1, 10)
```

```
7
```

Function randrange

- With ***randrange( )***, you can exclude the right-hand side of the interval, meaning the generated number always lies within [x, y) and will always be smaller than the right endpoint:
  - ***randrange(*** *Start*, *End*, *Step* ***)***

```
# Generate a random number between 1-9
random.randrange(1, 10)
```

```
9
```

```
# Generate a random number from 1, 3, 5, 7, 9
random.randrange(1, 10, 2)
```

```
3
```

Function choice

- Use the ***choice( )*** function to randomly select an item from a List or any sequence.
  - ***choice(*** *Start*, *End*, *Step* ***)***

```
MyList = ["A", "B", "C", "D", "E", "F", "G"]

# Generate a random item from the provided list
random.choice(MyList)
```

```
'C'
```

Function choices

- When you want to choose more than one element from the sequence randomly, then use ***choices( )***.
- Choices method introduced in python version 3.6 and it can repeat the elements. It is a random sample with replacement.

```
MyList = ["A", "B", "C"]

# Generate 2 items from the provided list (Can be duplicated)
random.choices(MyList, k=2)
```

```
['B', 'B']
```

Function sample

- When you want to pick multiple elements from a list or any sequence randomly, ***sample( )*** is the best choice.
  - ***sample(*** *population*, *k* ***)***

```
MyList = ["A", "B", "C", "D", "E", "F", "G"]

# Generate 2 samples from the provided list
random.sample(MyList, 2)
```

```
['A', 'F']
```

Function shuffles

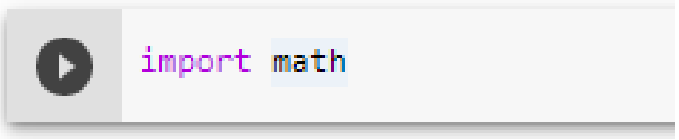
- The shuffle function, shuffles the elements in list in place, so they are in a random order.
  - ***shuffle(*** *list* ***)***

```
MyList = ["A", "B", "C", "D", "E", "F", "G"]

# Shuffles the original list to random order
random.shuffle(MyList)

# Print out the list
MyList
```

```
['E', 'A', 'D', 'B', 'F', 'G', 'C']
```

Module math

- The Python math module is an important feature designed to deal with mathematical operations. It comes packaged with the standard Python release and has been there from the beginning.



```
import math
```

- The return values are floats, detail function information can be found in:
  - https://docs.python.org/3/library/math.html

Constants of the math Module

- The Python math module offers a variety of predefined constants.
- The module includes several famous mathematical constants and important values:
  - Pi
  - Tau
  - Euler’s number
  - Infinity
  - Not a number (NaN)

Example

```
# Pi (π) is the ratio of a circle's circumference (c) to its diameter (d)
math.pi
```

```
3.141592653589793
```

```
# Tau (τ) is the ratio of a circle's circumference to its radius
math.tau
```

```
6.283185307179586
```

```
# Euler's number (e) is a constant that is the base of the natural logarithm
math.e
```

```
2.718281828459045
```

```
# Infinity representing something that is never-ending or boundless.
math.inf
```

```
inf
```

```
# A NaN value can be due to invalid inputs
# or numerical a variable has been corrupted by text characters or symbols.
math.nan
```

```
nan
```

Find the Ceiling Value with ceil

- The ***ceil( )*** function will return the smallest integer value that is greater than or equal to the given number.
- If the number is a positive or negative decimal, then the function will return the next integer value greater than the given value.

```
math.ceil(123.456)
```

```
124
```

```
math.ceil(-123.456)
```

```
-123
```

Find the Floor Value with floor

- The function ***floor( )*** will return the closest integer value that is less than or equal to the given number.
- This function behaves opposite to ceil().

```
math.floor(123.456)
```

```
123
```

```
math.floor(-123.456)
```

```
-124
```

Truncate Numbers with trunc

- When you get a number with a decimal point, you might want to keep only the integer part and eliminate the decimal part. The math module has a function called ***trunc( )*** which lets you do just that.
- Dropping the decimal value is a type of rounding. With ***trunc( )***, negative numbers are always rounded upward toward zero and positive numbers are always rounded downward toward zero.

Rounding Function Summary

- There are several functions performing rounding:
  - round()
  - math.ceil()
  - math.floor()
  - math.trunc()

Input	round()	ceil()	floor()	trunc()
56.7	57	57	56	56
-56.7	-57	-56	-57	-56

Arithmetic Functions

- The Python math module provides functions that are useful in number theory as well as in representation theory, a related field. These functions allow you to calculate a range of important values:

```
math.factorial(5)   #1 * 2 * 3 * 4 * 5 = 120
```

```
120
```

```
math.sqrt(49)   #Square root of 49 = 7
```

```
7.0
```

Find the Closeness with isclose

- The function *isclose( )* lets you set your own tolerance for closeness. It returns True if two numbers are within your established tolerance for closeness and otherwise returns False.
  - Relative Tolerance (rel_tol), is the maximum difference for being considered “close” relative to the magnitude of the input values. This is the percentage of tolerance. The default value is 1e-09 or 0.000000001.
  - Absolute Tolerance (abs_tol), is the maximum difference for being considered “close” regardless of the magnitude of the input values. The default value is 0.0.

Find the Closeness with isclose (cont.)

- The function ***isclose( )*** will return True when the following condition is satisfied:
  - abs(a-b) <= max(Relative Tolerance x max(abs(a), abs(b)), Absolute Tolerance).
- The functin uses the above expression to determine the closeness of two numbers. You can substitute your own values and observe whether any two numbers are close.

Find the Closeness with isclose (cont.)

- The numbers 6 and 7 aren't considered close because the relative tolerance is set for 9 d.p.

```
Python >>>
>>> math.isclose(6, 7)
False
```

- 6.999999999 and 7 are considered close under the same tolerance:

```
Python >>>
>>> math.isclose(6.999999999, 7)
True
```

Impact of Relative Tolerance

- If you set relative tolerance to 0.2, then 6 and 7 are considered close. This is because they are within 20% of each other.

```python
>>> math.isclose(6, 7, rel_tol=0.2)
True
```

Impact of Absolute Tolerance

- When you set the absolute tolerance to 1, the numbers 6 and 7 are close because the difference between them is equal to the absolute tolerance.
- However, in the second case, the difference between 6 and 7 is not less than or equal to the established absolute tolerance of 0.2.

Python >>>

```
>>> math.isclose(6, 7, abs_tol=1.0)
True
>>> math.isclose(6, 7, abs_tol=0.2)
False
```

Calculate the Power of a Number With pow

- Power functions have the following formula where the variable ***x*** is the base, the variable ***n*** is the power, and ***a*** can be any constant:
  - $f(x) = ax^n$

Python

```
>>> math.pow(2, 5)
32.0
>>> math.pow(5, 2.4)
47.59134846789696
```

Logarithmic Functions

- Logarithmic functions can be considered the inverse of exponential functions. They are denoted in the following form:
  - $f(x) = \log_a(x)$
- The natural logarithm of a number is its logarithm to the base of the mathematical constant e, or Euler's number:
  - $f(x) = \log_e(x)$

Python

```
>>> math.log(4)
1.3862943611198906
>>> math.log(3.4)
1.2237754316221157
```

Understand log2 and log10

- The Python math module provides two separate functions that let you calculate the log values to the base of 2 and 10:
  - ***log2( )*** is used to calculate the log value to the base 2.
  - ***log10( )*** is used to calculate the log value to the base 10.

Python

```
>>> math.log2(math.pi)
1.6514961294723187
>>> math.log(math.pi, 2)
1.651496129472319
```

Python

```
>>> math.log10(math.pi)
0.4971498726941338
>>> math.log(math.pi, 10)
0.4971498726941338
```

Calculate the Greatest Common Divisor

- The greatest common divisor (GCD) of two positive numbers is the largest positive integer that divides both numbers without a remainder.
  - For example, the GCD of 15 and 25 is 5. You can divide both 15 and 25 by 5 without any remainder. There is no greater number that does the same.
  - If you take 15 and 30, then the GCD is 15 because both 15 and 30 can be divided by 15 without a remainder.

Calculate the Sum of Iterables

- If you ever want to find the sum of the values of an iterable without using a loop, then ***fsum()*** is probably the easiest way to do so.
- You can use iterables such as arrays, tuples, or lists as input and the function returns the sum of the values.

```
MyList = [1, 2, 3, 4, 5]

# Sum all the element in the list
math.fsum( MyList)
```

```
15.0
```

Calculate Trigonometric Values

- Python math module provides very useful functions that let you perform trigonometric calculations

Function	Description
math.sin( )	Calculate the sine value of an angle
math.cos( )	Calculate the cosine value of an angle
math.tan( )	Calculate the tangent value of an angle
math.asin( )	Calculate the arc sine value of an angle
math.acos( )	Calculate the arc cosine value of an angle
math.atan( )	Calculate the arc tangent value of an angle
math.hypot( )	Calculate the hypotenuse of a triangle

Modules stat

- Python’s statistics is a built-in Python library for descriptive statistics. You can use it if your datasets are not too large or if you can’t rely on importing other libraries.
- The built-in Python statistics library has a relatively small number of the most important statistics functions.
- The official documentation is available in https://docs.python.org/3/library/statistics.html.

Function in Modules stat

Function	Description
harmonic_mean()	Calculates the harmonic mean (central location) of the given data
mean()	Calculates the mean (average) of the given data
median()	Calculates the median (middle value) of the given data
median_grouped()	Calculates the median of grouped continuous data
median_high()	Calculates the high median of the given data
median_low()	Calculates the low median of the given data
mode()	Calculates the mode (central tendency) of the given numeric or nominal data
pstdev()	Calculates the standard deviation from an entire population
stdev()	Calculates the standard deviation from a sample of data
pvariance()	Calculates the variance of an entire population
variance()	Calculates the variance from a sample of data

Module os

- Module os provides functions that interface with the underlying operating system
  - E.g. path submodule allows you to manipulate and find the properties of files and folders on the system

```
import os

# Returns the current working directory.
Current_Path = os.getcwd()
print(Current_Path)

# Return a list of the entries in the directory
os.listdir(Current_Path)
```

```
/home/nbuser/library/Lecture 3

['Chapter 3 Function.ipynb',
 'Chapter 3 Scope of Life.ipynb',
 '__pycache__',
 'Chapter 3 Module.ipynb',
 '.ipynb_checkpoints',
 'Financial.py']
```

Module datetime

- Datetime module supplies classes to work with date and time. These classes provide a number of functions to deal with dates, times and time intervals.
- Date and datetime are an object in Python, so when you manipulate them, you are actually manipulating objects and not string or timestamps.
- The official documentation is available in https://docs.python.org/3/library/datetime.html

```
import datetime
```

Main Classes in datetime module

- The datetime module are categorize into 6 main classes

Class	Description
date	An idealized naive date, assuming the current Gregorian calendar always was, and always will be, in effect. Its attributes are year, month and day.
time	An idealized time, independent of any particular day, assuming that every day has exactly 24x60x60 seconds. Its attributes are hour, minute, second, microsecond, and tzinfo.
datetime	Its a combination of date and time along with the attributes year, month, day, hour, minute, second, microsecond, and tzinfo.
timedelta	A duration expressing the difference between two date, time, or datetime instances to microsecond resolution.
tzinfo	It provides time zone information objects.
timezone	A class that implements the tzinfo abstract base class as a fixed offset from the UTC

Date Attributes

- This class contains the following attributes

Attributes	Description
year	Returns the year from a date object
month	Returns the month from a date object
day	Returns the day from a date object
hour	Returns the hour from a date object
minutes	Returns the minutes from a date object
second	Returns the second from a date object

Get Current Date Time

- By using the function ***today( )*** and ***now()***, we can obtain the current date and time.

```
# Get Current Date
MyDate = datetime.date.today()

print(MyDate)
```

```
2020-10-13
```

```
# Get Current Date and Time
MyDate = datetime.datetime.now()

print(MyDate)
```

```
2020-10-13 15:28:22.924162
```

Weekday Number

- Weekday number is useful for arrays whose index is dependent on the day of the week.
- The function ***weekday( )*** return the day of the week, where 0 denotes Monday.
- The function ***isoweekday( )*** return the day of the week as an integer, where Monday is 1 and Sunday is 7.

```
# Print weekday number (Monday = 0)
MyDate.weekday()
```

```
1
```

```
# Print weekday number (Monday = 1)
MyDate.isoweekday()
```

```
2
```

Weekday Number (cont.)

- Comparison between ***weekday( )*** and ***isoweekday( )***.

Day	weekday( )	isoweekday( )
Monday	0	1
Tuesday	1	2
Wednesday	2	3
Thursday	3	4
Friday	4	5
Saturday	5	6
Sunday	6	7

Formatting Date

- The datetime object has a method for formatting date objects into readable strings.
- The method is called ***strftime( )***, and takes one parameter, format, to specify the format of the returned string.
- The detail strftime reference can be obtained from http://strftime.org

```
# Format the date: dd / mm / yyyy
MyDate.strftime("%d / %m / %Y")
```

```
'31 / 12 / 2099'
```

Reference of Legal Format Code

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%

Create Date Object

- The function ***date( )*** is a constructor of the date class.
- The constructor takes three arguments: year, month and day.

```
# Create a date object
MyDate = datetime.date(2099,12,31)

print(MyDate)
```

```
2099-12-31
```

Convert String to Date

- The ***strptime( )*** method creates a datetime object from the given string.
- You cannot create datetime object from every string. The string needs to be in a certain format.

```
# Convert the string to date
MyDate = datetime.datetime.strptime("06-06-2018", '%m-%d-%Y')

print(MyDate)
```

```
2018-06-06 00:00:00
```

Create Custom Module

- Modules are quite easy to create. They are simply Python files, like your regular scripts.
- To create a module, write one or more functions in a text file, then save it with a .py extension.

```
def CalculateInterest(Capital, Rate):
    Interest = Capital * Rate / 100
    return Interest
```

- To use the module, just import the filename, and call the function

```
import Financial

print(Financial.CalculateInterest(10000, 5))
```

```
500.0
```

Top Python Libraries for Data Science