



Python Programming

Hyuk Lim

Energy AI Track
Korea Institute of Energy Technology

Contents

- Introduction
- Using Python as a Calculator
- Replit Exercise

Introduction

<https://hlim.kentech.ac.kr>

3

Top Programming Languages

Rank	Language	Type	Score
1	Python	Web	100.0
2	Java	Web	95.4
3	C	System	94.7
4	C++	System	92.4
5	JavaScript	Web	88.1
6	C#	Web	82.4
7	R	Web	81.7
8	Go	Web	77.7
9	HTML	Web	75.4
10	Swift	Mobile	70.4

<https://spectrum.ieee.org/top-programming-languages-2021>

STACK OVERFLOW SURVEY

#	2019	2018	2017	2016	2015
1	JavaScript 69.7%	JavaScript 71.5%	JavaScript 66.7%	JavaScript 55.4%	JavaScript 54.4%
2	HTML/CSS 63.1%	HTML 69.4%	SQL 53.7%	SQL (or SQL Server) 49.1%	SQL 48.0%
3	SQL 56.5%	CSS 66.2%	Java 38.3%	Java 36.3%	Java 37.4%
4	Python 39.4%	SQL 58.5%	C# 36.7%	C# 30.9%	C# 31.6%
5	Java 39.2%	Java 45.4%	Python 27.6%	PHP 25.9%	PHP 29.7%
6	Bash/Shell/PowerShell 37.9%	Bash/Shell 40.4%	PHP 27.2%	Python 24.9%	Python 23.8%
7	C# 31.9%	Python 37.9%	C++ 19.3%	C++ 19.4%	C++ 20.6%
8	PHP 25.8%	C# 35.3%	C 15.4%	C 15.5%	C 16.4%
9	TypeScript 23.5%	PHP 31.4%	TypeScript 11.3%	Node.js 17.2%	Node.js 13.3%
10	C++ 20.4%	C++ 24.6%	Ruby 9.5%	AngularJS 17.9%	AngularJS 13.3%
11	C 17.3%	C 22.1%	Objective-C 7.3%	Ruby 8.9%	Ruby 8.0%
12	Ruby 8.9%	TypeScript 18.3%	Swift 6.9%	Objective-C 6.5%	Objective-C 7.8%
13	Go 8.8%	Ruby 10.3%	VB.NET 6.1%		
14	Swift 6.8%	Swift 8.3%	Go 4.6%		
15	Kotlin 6.6%	Objective-C 7.3%	Perl 4.1%		

■ 5 years in chart ■ 4 years ■ 3 years ■ 2 years ■ 1 year

<https://light-it.net/blog/most-popular-programming-languages-of-the-last-decade-and-best-choices/>

<https://hlim.kentech.ac.kr>

4

"Give a man a fish and you feed him for a day. Teach a man to fish and you feed him for a lifetime."

- Chinese Proverb

10000 HOUR RULE AND LANGUAGE LEARNING



*answers on your
What and How*

"Give a man a fish and you feed him for a day. Teach a man to fish and you feed him for a lifetime."

- Chinese Proverb

10000 HOUR RULE AND LANGUAGE LEARNING



*answers on your
What and How*

<https://www.python.org/doc/>
<https://docs.oracle.com/javase/8/>

<https://swexpertacademy.com/>
<https://leetcode.com/>
<https://www.hackerrank.com/>

<https://www.python.org/doc/>

Python 3.x Resources

- Browse Python 3.10.2 Documentation - (Module Index)
 - What's new in Python 3.10
 - Tutorial
 - Library Reference
 - Language Reference
 - Extending and Embedding
 - Python/C API
 - Using Python
 - Python HOWTOs
 - Glossary
- Search the online docs
- Download Current Documentation (multiple formats are available, including typeset versions for printing.)

The Python Standard Library

While The Python Language Reference describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually include the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

In addition to the standard library, there is a growing collection of several thousand components (from individual programs and modules to packages and entire application development frameworks), available from the Python Package Index.

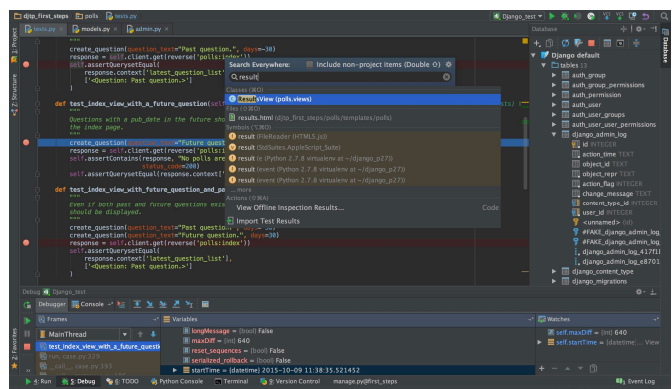
- Introduction
 - Notes on availability
- Built-in Functions
- Built-in Constants
 - Constants added by the site module
- Built-in Types
 - Truth Value Testing
 - Boolean Operations — and, or, not
 - Comparisons
 - Numeric Types — int, float, complex
 - Iterator Types
 - Sequence Types — list, tuple, range
 - Text Sequence Type — str
 - Binary Sequence Types — bytes, bytearray, memoryview
 - Set Types — set, frozenset
 - Mapping Types — dict
 - Context Manager Types
 - Type Annotation Types — Generic Alias, Union
 - Other Built-in Types
 - Special Attributes

<https://hlim.kentech.ac.kr>

7

Writing Python Codes


- A plain text editor
 - Windows Notepad
- An Integrated Development Environment (IDE)
 - PyCharm
 - Spyder ...
- The Jupyter Notebook

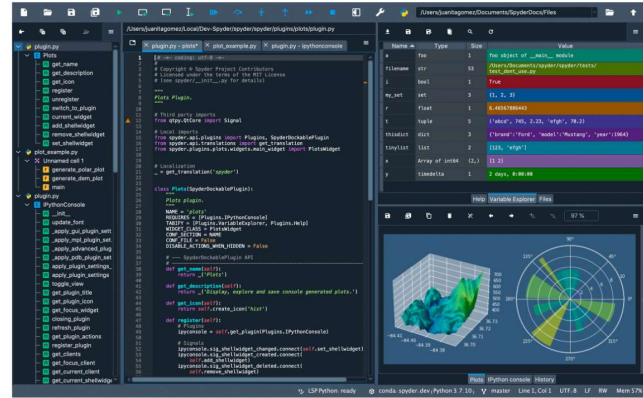


<https://hlim.kentech.ac.kr>

8

Writing Python Codes


- A plain text editor
 - Windows Notepad
- An Integrated Development Environment (IDE)
 - PyCharm
 - Spyder (Anaconda) 
- The Jupyter Notebook

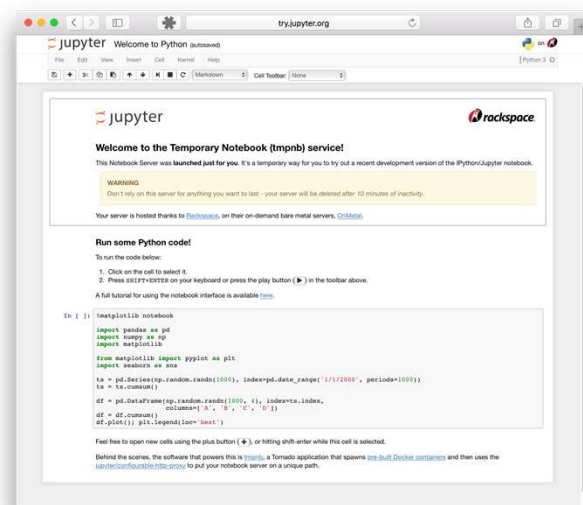


<https://hl.m.kentech.ac.kr>

9

Writing Python Codes

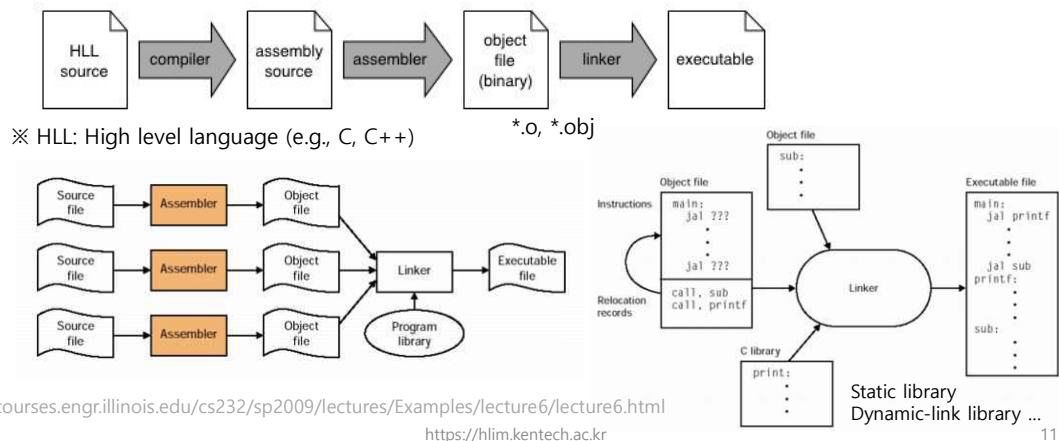
- A plain text editor
 - Windows Notepad
- An Integrated Development Environment (IDE)
 - PyCharm
 - Spyder ...
- The Jupyter Notebook
 - Google Colab 



10

Python Interpreter vs Compiler

- Compiler, assembler, linker → Executable file (*.out, *.exe, ...)



11

Python Interpreter vs Compiler

- In computer science, an interpreter is **a computer program that directly executes instructions written in a programming or scripting language**, without requiring them previously to have been compiled into a machine language program.
- Examples: Python, MATLAB, Ruby, ...

- Augument passing vs interactive mode
 - Agument passing: python.exe your_file.py -h
 - Interactive mode
 - Primary prompt (>>>) and secondary prompt (...)

```
$ python3.10
Python 3.10 (default, June 4 2019, 09:25:04)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license()" for more information.
>>>
Continuation lines are needed when entering a multi-line construct. As an example, take a look at this if statement.
>>> the_world_is_flat = True
>>> if the_world_is_flat:
...     print("Be careful not to fall off!")
...
Be careful not to fall off!
```

Python/Doc/Tutorial/2.1.2

<https://hlim.kentech.ac.kr>

12

Using Python as a Calculator

<https://hlim.kentech.ac.kr>

13

Using Python as a Calculator

3.1.1. Numbers

The interpreter acts as a simple calculator: you can type an expression at it and it will write the value. Expression syntax is straightforward: the operators `+`, `-`, `*` and `/` work just like in most other languages (for example, Pascal or C); parentheses `()` can be used for grouping. For example:

```
>>> 2 + 2
4
>>> 50 - 5*6
20
>>> (50 - 5*6) / 4
5.0
>>> 8 / 5 # division always returns a floating point number
1.6
```

The integer numbers (e.g. 2, 4, 20) have type `int`, the ones with a fractional part (e.g. 5.0, 1.6) have type `float`. We will see more about numeric types later in the tutorial.

<https://hlim.kentech.ac.kr>

14

Division (/) always returns a float. To do [floor division](#) and get an integer result (discarding any fractional result) you can use the // operator; to calculate the remainder you can use %:

```
>>> 17 / 3 # classic division returns a float
5.666666666666667
>>>
>>> 17 // 3 # floor division discards the fractional part
5
>>> 17 % 3 # the % operator returns the remainder of the division
2
>>> 5 * 3 + 2 # floored quotient * divisor + remainder
17
```

With Python, it is possible to use the ** operator to calculate powers [1]:

```
>>> 5 ** 2 # 5 squared
25
>>> 2 ** 7 # 2 to the power of 7
128
```

The equal sign (=) is used to assign a value to a variable. Afterwards, no result is displayed before the next interactive prompt:

```
>>> width = 20
>>> height = 5 * 9
>>> width * height
900
```

<https://hlim.kentech.ac.kr>

15

If a variable is not "defined" (assigned a value), trying to use it will give you an error:

```
>>> n # try to access an undefined variable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

There is full support for floating point; operators with mixed type operands convert the integer operand to floating point:

```
>>> 4 * 3.75 - 1
14.0
```

In interactive mode, the last printed expression is assigned to the variable `_`. This means that when you are using Python as a desk calculator, it is somewhat easier to continue calculations, for example:

```
>>> tax = 12.5 / 100
>>> price = 100.50
>>> price * tax
12.5625
>>> price + _
113.0625
>>> round(_, 2)
113.06
```

This variable should be treated as read-only by the user. Don't explicitly assign a value to it — you would create an independent local variable with the same name masking the built-in variable with its magic behavior.

In addition to `int` and `float`, Python supports other types of numbers, such as `Decimal` and `Fraction`. Python also has built-in support for [complex numbers](#), and uses the `j` or `J` suffix to indicate the imaginary part (e.g. `3+5j`).

<https://hlim.kentech.ac.kr>

16

Python Operators

<https://docs.python.org/3/library/operator.html#mapping-operators-to-functions>

Operation	Syntax	Function
Addition	<code>a + b</code>	<code>add(a, b)</code>
Concatenation	<code>seq1 + seq2</code>	<code>concat(seq1, seq2)</code>
Containment Test	<code>obj in seq</code>	<code>contains(seq, obj)</code>
Division	<code>a / b</code>	<code>truediv(a, b)</code>
Division	<code>a // b</code>	<code>floordiv(a, b)</code>
Bitwise And	<code>a & b</code>	<code>and_(a, b)</code>
Bitwise Exclusive Or	<code>a ^ b</code>	<code>xor(a, b)</code>
Bitwise Inversion	<code>~ a</code>	<code>invert(a)</code>
Bitwise Or	<code>a b</code>	<code>or_(a, b)</code>
Exponentiation	<code>a ** b</code>	<code>pow(a, b)</code>
Identity	<code>a is b</code>	<code>is_(a, b)</code>
Identity	<code>a is not b</code>	<code>is_not(a, b)</code>
Indexed Assignment	<code>obj[k] = v</code>	<code>setitem(obj, k, v)</code>
Indexed Deletion	<code>del obj[k]</code>	<code>delitem(obj, k)</code>
Indexing	<code>obj[k]</code>	<code>getitem(obj, k)</code>

and more ...

<https://hlim.kentech.ac.kr>

17

Mathematical Functions

<https://docs.python.org/3/library/math.html>

- Python Standard Library – **math**
 - Number-theoretic and representation functions
 - Power and logarithmic functions
 - Trigonometric functions
 - Angular conversion
 - Hyperbolic functions
 - ...
- How to use it?
 - `>>> import math`
 - `>>> x = math.ceil(2.3)`

<https://hlim.kentech.ac.kr>

18

Python Functions

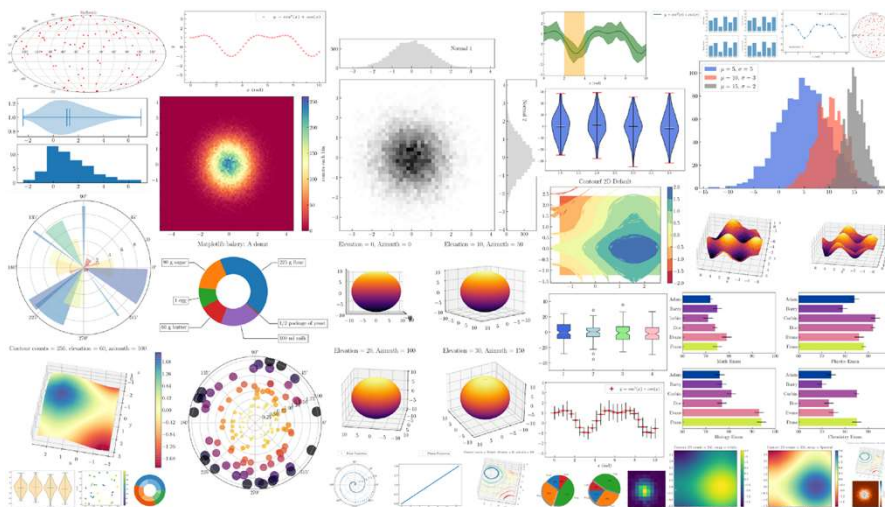
- Python Standard Library
- Built-in functions
<https://docs.python.org/3/library/functions.html>
- Third party package
 - Installation required
c:W> pip install package_name

Built-in Functions			
A abs() alter() all() any() anext() ascii()	E enumerate() eval() exec()	L len() list() locals()	R range() repr() reversed() round()
B bin() bool() breakpoint() bytearray() bytes()	F filter() float() format() frozenset()	M map() max() memoryview() min()	S set() setattr() slice() sorted() staticmethod() str() sum() super()
C callable() chr() classmethod() compile() complex()	G getattr() globals()	N next()	O object() oct() open() ord()
D delattr() dict() dir() divmod()	H hasattr() hash() help() hex()	P pow() print() property()	T tuple() type()
	I id() input() int() isinstance() issubclass() iter()	V vars()	Z zip()
			_ __import__()

<https://hlim.kentech.ac.kr>

19

matplotlib: Visualization with Python



<https://hlim.kentech.ac.kr>

20

Height of a Ball in Vertical Motion

$$y(t) = v_0 t - \frac{1}{2} g t^2$$

where

- y is the height (position) as function of time t
- v_0 is the initial velocity at $t = 0$
- g is the acceleration of gravity

For $v_0 = 5$, $g = 9.91$, and $t = 0.6$, what is the height?

Using Python as a Calculator

3.1.2. Strings

Besides numbers, Python can also manipulate strings, which can be expressed in several ways. They can be enclosed in single quotes ('...') or double quotes ("...") with the same result [2]. `\` can be used to escape quotes:

```
>>> 'spam eggs' # single quotes
'spam eggs'
>>> 'doesn\'t' # use \' to escape the single quote...
"doesn't"
>>> "doesn't" # ...or use double quotes instead
"doesn't"
>>> '"Yes," they said.'
'"Yes," they said.'
>>> "\"Yes,\" they said."
'"Yes," they said.'
>>> '"Isn\'t," they said.'
'"Isn't," they said.'
```

In the interactive interpreter, the output string is enclosed in quotes and special characters are escaped with backslashes. While this might sometimes look different from the input (the enclosing quotes could change), the two strings are equivalent. The string is enclosed in double quotes if the string contains a single quote and no double quotes, otherwise it is enclosed in single quotes. The `print()` function produces a more readable output, by omitting the enclosing quotes and by printing escaped and special characters:

Strings can be *indexed* (subscripted), with the first character having index 0. There is no separate character type; a character is simply a string of size one:

```
>>> word = 'Python'
>>> word[0] # character in position 0
'P'
>>> word[5] # character in position 5
'n'
```

Indices may also be negative numbers, to start counting from the right:

```
>>> word[-1] # last character
'n'
>>> word[-2] # second-last character
'o'
>>> word[-6]
'P'
```

Note that since -0 is the same as 0, negative indices start from -1.

In addition to indexing, *slicing* is also supported. While indexing is used to obtain individual characters, *slicing* allows you to obtain substring:

```
>>> word[0:2] # characters from position 0 (included) to 2 (excluded)
'Py'
>>> word[2:5] # characters from position 2 (included) to 5 (excluded)
'tho'
```

Slice indices have useful defaults; an omitted first index defaults to zero, an omitted second index defaults to the size of the string being sliced.

```
>>> word[:2] # character from the beginning to position 2 (excluded)
'Py'
>>> word[4:] # characters from position 4 (included) to the end
'on'
>>> word[-2:] # characters from the second-last (included) to the end
'on'
```

Note how the start is always included, and the end always excluded. This makes sure that `s[:i] + s[i:]` is always equal to `s`:

```
>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

input()

`input([prompt])`

If the *prompt* argument is present, it is written to standard output without a trailing newline. The function then reads a line from input, converts it to a string (stripping a trailing newline), and returns that. When EOF is read, `EOFError` is raised. Example:

```
>>> s = input('--> ')
--> Monty Python's Flying Circus
>>> s
'Monty Python's Flying Circus'
```

If the `readline` module was loaded, then `input()` will use it to provide elaborate line editing and history features.

Raises an auditing event `builtins.input` with argument `prompt` before reading input.

Raises an auditing event `builtins.input/result` with the result after successfully reading input.

```
>>> input_string = input('Please, enter a number')
>>> x = int(input_string)
```

print() and str.format()

7.1. Fancier Output Formatting

So far we've encountered two ways of writing values: *expression statements* and the `print()` function. (A third way is using the `write()` method of file objects; the standard output file can be referenced as `sys.stdout`. See the Library Reference for more information on this.)

Often you'll want more control over the formatting of your output than simply printing space-separated values. There are several ways to format output.

- To use *formatted string literals*, begin a string with `f` or `F` before the opening quotation mark or triple quotation mark. Inside this string, you can write a Python expression between `{` and `}` characters that can refer to variables or literal values.

```
>>> year = 2016
>>> event = 'Referendum'
>>> f'Results of the {year} {event}'
'Results of the 2016 Referendum'
```

- The `str.format()` method of strings requires more manual effort. You'll still use `{}` and `}` to mark where a variable will be substituted and can provide detailed formatting directives, but you'll also need to provide the information to be formatted.

```
>>> yes_votes = 42_572_654
>>> no_votes = 43_132_495
>>> percentage = yes_votes / (yes_votes + no_votes)
>>> '{:-9} YES votes  {:.2%}'.format(yes_votes, percentage)
' 42572654 YES votes  49.67%'
```

7.1.2. The String `format()` Method

Basic usage of the `str.format()` method looks like this:

```
>>> print('We are the {} who say "{}!"'.format('knights', 'Ni'))
We are the knights who say "Ni!"
```

The brackets and characters within them (called format fields) are replaced with the objects passed into the `str.format()` method. A number in the brackets can be used to refer to the position of the object passed into the `str.format()` method.

```
>>> print('{0} and {1}'.format('spam', 'eggs'))
spam and eggs
>>> print('{1} and {0}'.format('spam', 'eggs'))
eggs and spam
```

<https://hlim.kentech.ac.kr>

27

If keyword arguments are used in the `str.format()` method, their values are referred to by using the name of the argument.

```
>>> print('This {food} is {adjective}.'.format(
...     food='spam', adjective='absolutely horrible'))
This spam is absolutely horrible.
```

Positional and keyword arguments can be arbitrarily combined:

```
>>> print('The story of {0}, {1}, and {other}.'.format('Bill', 'Manfred',
...     other='Georg'))
The story of Bill, Manfred, and Georg.
```

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{0}, {0}, {0}'.format('a', 'b', 'c') # 3, 1+ only
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'
>>> '{2}, {1}, {0}'.format(*'abc') # unpacking argument sequence
'c, b, a'
>>> '{0}{1}{0}'.format('abra', 'cad') # arguments' indices can be repeated
'abracadabra'
```

<https://docs.python.org/3/library/string.html#formatstrings>

Accessing arguments by name:

```
>>> 'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N', longitude='-115.81W')
'Coordinates: 37.24N, -115.81W'
>>> coord = {'latitude': '37.24N', 'longitude': '-115.81W'}
>>> 'Coordinates: {latitude}, {longitude}'.format(**coord)
'Coordinates: 37.24N, -115.81W'
```

<https://hlim.kentech.ac.kr>

28

Using the comma as a thousands separator:

```
>>> '{:,}'.format(1234567890)
'1,234,567,890'
```

Expressing a percentage:

```
>>> points = 19
>>> total = 22
>>> 'Correct answers: {:.2%}'.format(points/total)
'Correct answers: 86.36%'
```

Using type-specific formatting:

```
>>> import datetime
>>> d = datetime.datetime(2010, 7, 4, 12, 15, 58)
>>> '{:%Y-%m-%d %H:%M:%S}'.format(d)
'2010-07-04 12:15:58'
```

Exercise 1)

Write a Python code for the previous example 'Height of a Ball in Vertical Motion'

Filename: 'ex1_StudentID'.py (e.g., ex1_123456.py)

- Print a short descript for your program
- Ask the initial velocity and time
- Compute the position using the formula.
- Print the result value rounded to 2 decimal places.

※ Upload your file to LMS

Exercise 2)

Write a Python code for finding the roots of a second-order polynomial equation.

Filename: 'ex2_StudentID'.py (e.g., ex2_123456.py)

- Ask the coefficient values
- Compute the roots
- Print the result value rounded to 3 decimal places.

✂ Upload your file to LMS

Lists

3.1.3. Lists

Python knows a number of *compound* data types, used to group together other values. The most versatile is the *list*, which can be written as a list of comma-separated values (items) between square brackets. Lists might contain items of different types, but usually the items all have the same type.

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares
[1, 4, 9, 16, 25]
```

Like strings (and all other built-in [sequence](#) types), lists can be indexed and sliced:

```
>>> squares[0] # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:] # slicing returns a new list
[9, 16, 25]
```


All slice operations return a new list containing the requested elements. This means that the following slice returns a [shallow copy](#) of the list:

```
>>> squares[:]  
[1, 4, 9, 16, 25]
```

Lists also support operations like concatenation:

```
>>> squares + [36, 49, 64, 81, 100]  
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Unlike strings, which are [immutable](#), lists are a [mutable](#) type, i.e. it is possible to change their content:

```
>>> cubes = [1, 8, 27, 65, 125] # something's wrong here  
>>> 4 ** 3 # the cube of 4 is 64, not 65!  
64  
>>> cubes[3] = 64 # replace the wrong value  
>>> cubes  
[1, 8, 27, 64, 125]
```

You can also add new items at the end of the list, by using the `append()` *method* (we will see more about *methods* later):

```
>>> cubes.append(216) # add the cube of 6  
>>> cubes.append(7 ** 3) # and the cube of 7  
>>> cubes  
[1, 8, 27, 64, 125, 216, 343]
```

Assignment to slices is also possible, and this can even change the size of the list or clear it entirely:

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']  
>>> letters  
['a', 'b', 'c', 'd', 'e', 'f', 'g']  
>>> # replace some values  
>>> letters[2:5] = ['C', 'D', 'E']  
>>> letters  
['a', 'b', 'C', 'D', 'E', 'f', 'g']  
>>> # now remove them  
>>> letters[2:5] = []  
>>> letters  
['a', 'b', 'f', 'g']  
>>> # clear the list by replacing all the elements with an empty list  
>>> letters[:] = []  
>>> letters  
[]
```

The built-in function `len()` also applies to lists:

```
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)
4
```

It is possible to nest lists (create lists containing other lists), for example:

```
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x
[['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]
['a', 'b', 'c']
>>> x[0][1]
'b'
```

Replit Exercise



- Please, signup now!
- Join the team for VC Spring 2022.
- Solve two problems with your friend
 - Min and Max
 - Min Sum and Max Sum

Any Question?

hlim@kentech.ac.kr