

ACS Education 2nd

Programming



Index

- Fundamentals of programming theory
- C programming
- Python programming
- Web programming

01

Fundamentals of programming theory

- Programming overview
- Understanding the programming language

Programming overview

Programming implications

Programming is the skill of implementing abstract algorithms into practical computer programs using a specific programming language.

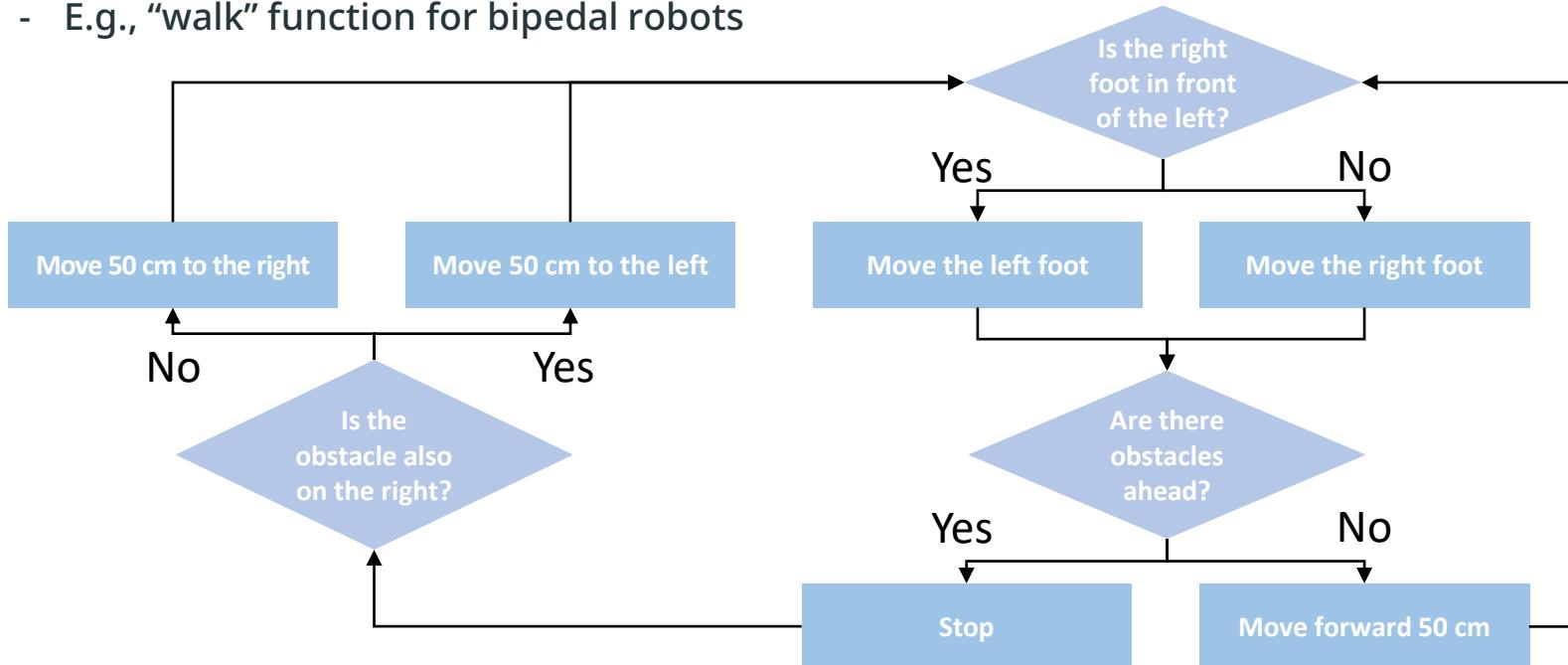
- Academic definition of programming
 - The process of designing and building an executable computer program to achieve a specific result
 - Analyzing, creating algorithms, profiling the accuracy and resource consumption of algorithms, and implementing algorithms
- A simple definition of programming
 - Literally, the act of creating a program itself
 - Program : a set of commands given to a computer for a specific purpose
 - Type of communication that asks a computer to perform actions
 - Need to clearly understand the problem and then define an appropriate solution (i.e., computational thinking)
 - Task of describing requirements with enough precision and detail that a machine that knows only 0s and 1s can execute them

Programming overview

Programmatic approach

When most people hear the word programming, they think of something difficult and esoteric. There is no doubt that it will be difficult for those who are new to programming. You will be able to communicate with your computer as you become familiar with it.

- Computational Thinking (CT)
 - Ability to simplify complex problems and solve them logically and efficiently
 - E.g., "walk" function for bipedal robots



Programming overview

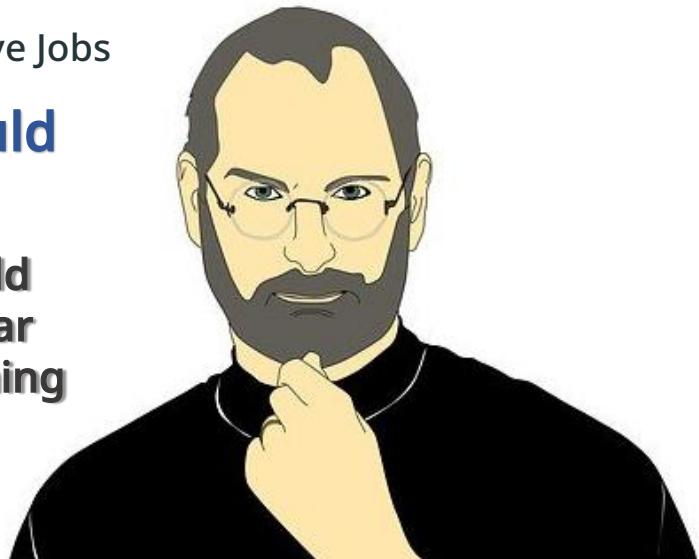
Why you should learn to program

When most people hear the word programming, they think of something difficult and esoteric. There is no doubt that it will be difficult for those who are new to programming. You will be able to communicate with your computer as you become familiar with it.

- Improve reasoning and problem-solving skills
- Improve collaboration and communication skills
- Programming skills aren't just for developers anymore; they're needed in a variety of fields.
- Excerpts from a 1995 interview with Apple founder Steve Jobs

"I think everybody in this country should learn how to program a computer."

"I view computer science as a liberal art. It should be something that everybody learns. Takes a year in their life, one of the courses they take is learning how to program."



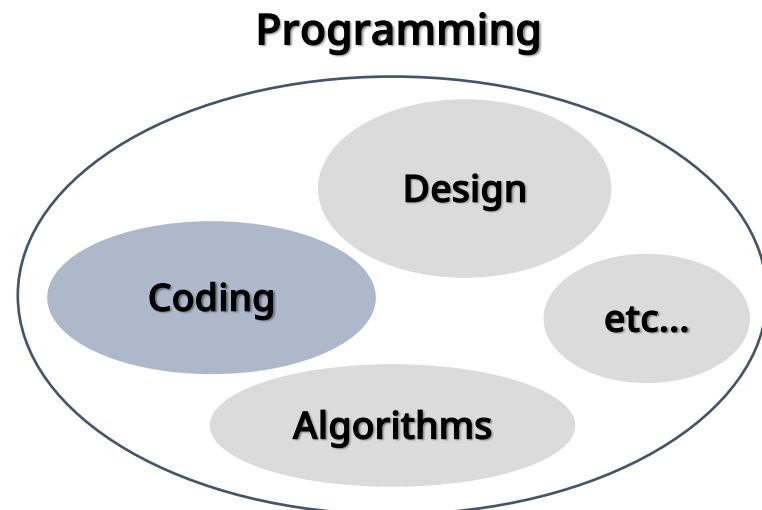
Sources : Schoolofweb.net (2019-02-11, Korean), Steve Jobs: The Lost Interview (2012-05-11)

Programming overview

Programming vs. coding

Coding and programming are two of the most important approaches in the software development industry. While coding and programming are often considered synonymous, they are significantly different.

- Coding
 - Work of translating algorithms into commands in a chosen programming language and then writing them.
 - Part of programming because it is used in the early stages of programming
- Differences in dictionary definitions
 - Programming
 - Act or process of writing a computer program
 - Tasks of designing how to write a program, coding it, fixing errors, among many other things
 - Coding
 - Method of converting information into a form suitable for computational manipulation



Programming overview

Programmer's dilemma



Programming overview

Programmer's mindset

When most people hear the word programming, they think of something difficult and esoteric. There is no doubt that it will be difficult for those who are new to programming. You will be able to communicate with your computer as you become familiar with it.

- Read the following Internet meme and think about why this programmer bought a lot of milk.

Being a Programmer

Mom said: "Please go to the shop and buy 1 bottle of milk. If they have eggs, bring 6"

I came back with 6 bottle of milk.



She said: "Why the hell did you buy 6 bottles of milk?"

I said: ?

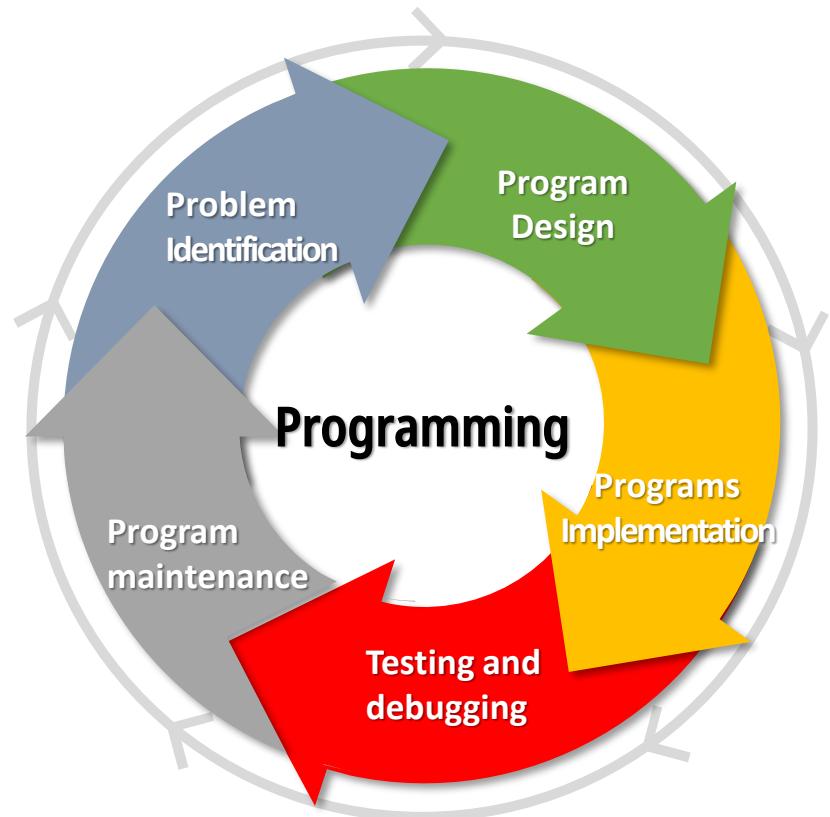
WebDevelopersNotes.com

Programming overview

Programming sequence

In the same way that constructing a building requires a blueprint, designing a program also requires careful consideration of its features and implementation.

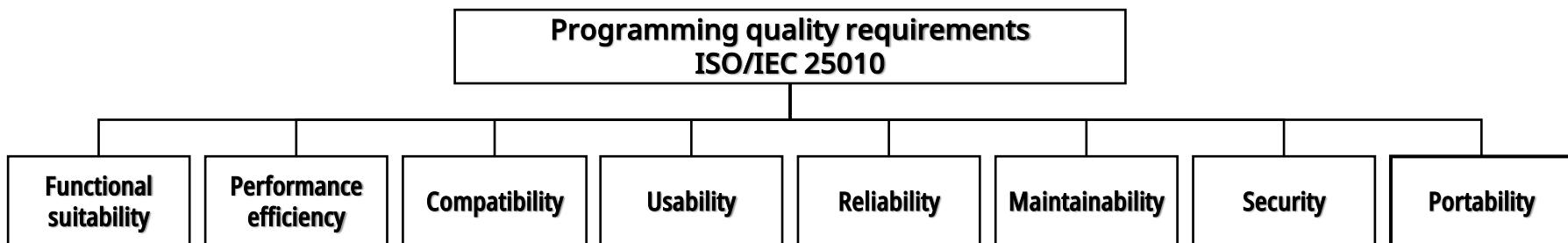
- Design process
 - Problem identification + program design
 - Create a program blueprint
- Implementation process
 - Program implementation + testing and debugging
 - Execute coding directly from the blueprint
 - Find and correct errors in the programs you write
- Follow-up process
 - Program maintenance
 - Make the necessary changes or additions



Programming overview

Things to consider when programming

- SW quality requirements - ISO/IEC 25010 (2011)
 - Functional suitability : capability to provide functions that meet stated and implied needs
 - Performance efficiency : capability to perform with appropriate use of resources and specified time
 - Compatibility : capability to perform functions sharing common conditions with other systems
 - Usability : capability to make it easy for users to understand and learn
 - Reliability : capability to perform intended functions and tasks under specified conditions without failure
 - Maintainability : capability of the software to be easily modified and changed
 - Security : capability to protect information and data
 - Portability : capability to operate in a variety of supported environments



Sources : www.iso.org

Programming overview

Things to consider when programming

- Source code readability
 - Make it easy for program developers to understand the purpose, control flow, and source code action
 - Influence the above quality aspects, including portability, usability, and maintainability
 - Readability factors
 - Indentation or spacing styles, comments (annotations), naming of objects (e.g., variables)
- Algorithm complexity
 - Affect resource usage, such as runtime and memory consumption; the less complex the algorithm, the more efficient it is.
 - Quantify the algorithm complexity using asymptotic notation
 - Asymptotic notation
 - Best cases : big Omega (Ω) notation
 - Average cases : big Theta (Θ) notation
 - Worst cases : big O notation

Programming overview

Things to consider when programming

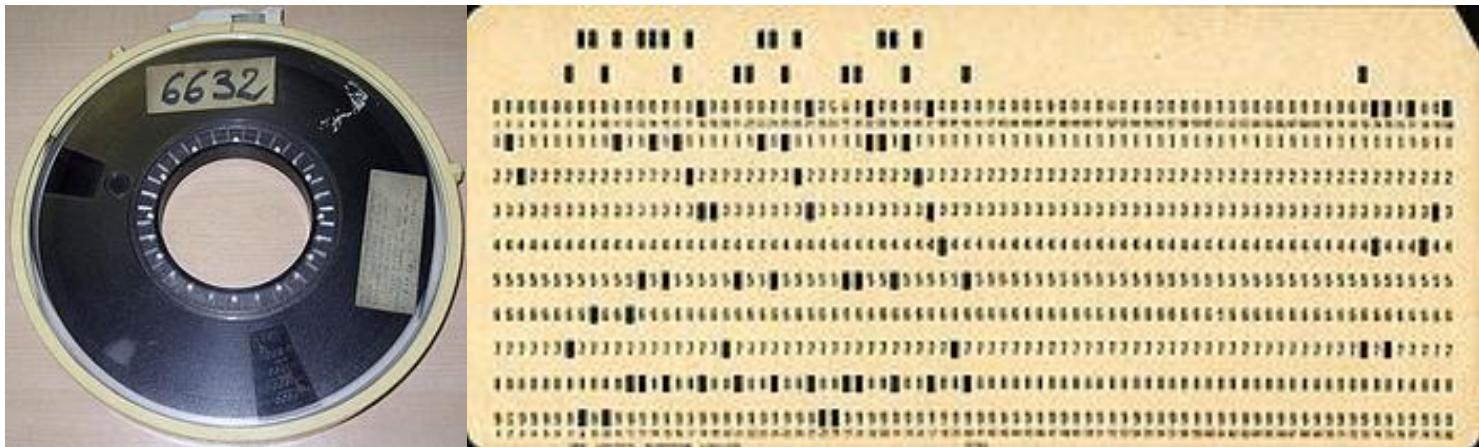
- Programming methodology
 - Software Development Life Cycle (SDLC)
 - Prescribe the working style at each stage and the documentation format
 - Requirements specification - analysis - design - development - testing - maintenance
 - Types of development methodologies
 - Structured, object-oriented, Component-Based Development (CBD)
- Language selection
 - Choose a language to program in
 - When developing a specific application
 - E.g., COBOL, Fortran, C, etc.
 - When writing multiple applications and making a choice based on compatibility
 - E.g., C++, JAVA, Python, etc.

Understanding the programming language

Programming language overview

The history of programming languages has evolved from early mechanical computer documentation to modern software development tools. It has led to the creation of advanced programming languages that use accessible syntax to convey commands.

- History of programming languages
 - First Generation (programming) Languages (1GLs)
 - Program in decimal or binary format
 - Absolute machine language
 - Read from a perforated card or a magnetic tape, or by operating switches on the front panel of a computer



Source: Wikipedia

Understanding the programming language

Programming language overview

The history of programming languages has evolved from early mechanical computer documentation to modern software development tools. It has led to the creation of advanced programming languages that use accessible syntax to convey commands.

- History of programming languages
 - Second Generation (programming) Languages (2GLs)
 - Assembly language
 - 1:1 correspondence with machine language, no compiler needed
 - Fast equivalent to machine language
 - E.g., x86 CPU machine language (10110000 01100001) -> assembly language (mov al, 061h)
 - Third Generation (programming) Languages (3GLs)
 - High-level programming languages currently in common use
 - Converted by compiler or interpreter (machine independent)
 - C/C++, C#, Java, BASIC

Understanding the programming language

Programming language overview

A programming language is a formal tool for writing software to run on computer systems. The more advanced the language, the more closely it resembles human language.

- Definition of a programming language
 - A system of program notation for writing computing performance or algorithms
 - Key characteristics of programming languages
 - Features and goals
 - Languages designed for writing computer programs
 - Perform computations or algorithms on a computer system and control external devices such as printers, disk drives, robots
 - Abstraction
 - Define and manipulate data structures or control execution flow
 - Expressiveness
 - The range of ideas that can be expressed and communicated in a programming language
 - The more expressive the greater the variety of ideas earned.

Understanding the programming language

Programming language overview

A programming language is a formal tool for writing software to run on computer systems. The more advanced the language, the more closely it resembles human language.

- Elements of a programming language

- Syntax

- Define the appearance, form and structure of a program → ensure the grammar of a language
 - Word : determine how characters form a token
 - Phrase : Determine how tokens form a phrase
 - Context : determine which objects or variable names are being referred to

- Semantics

- Define the meaning of a program → ensure that a sentence has a valid meaning

- Type system

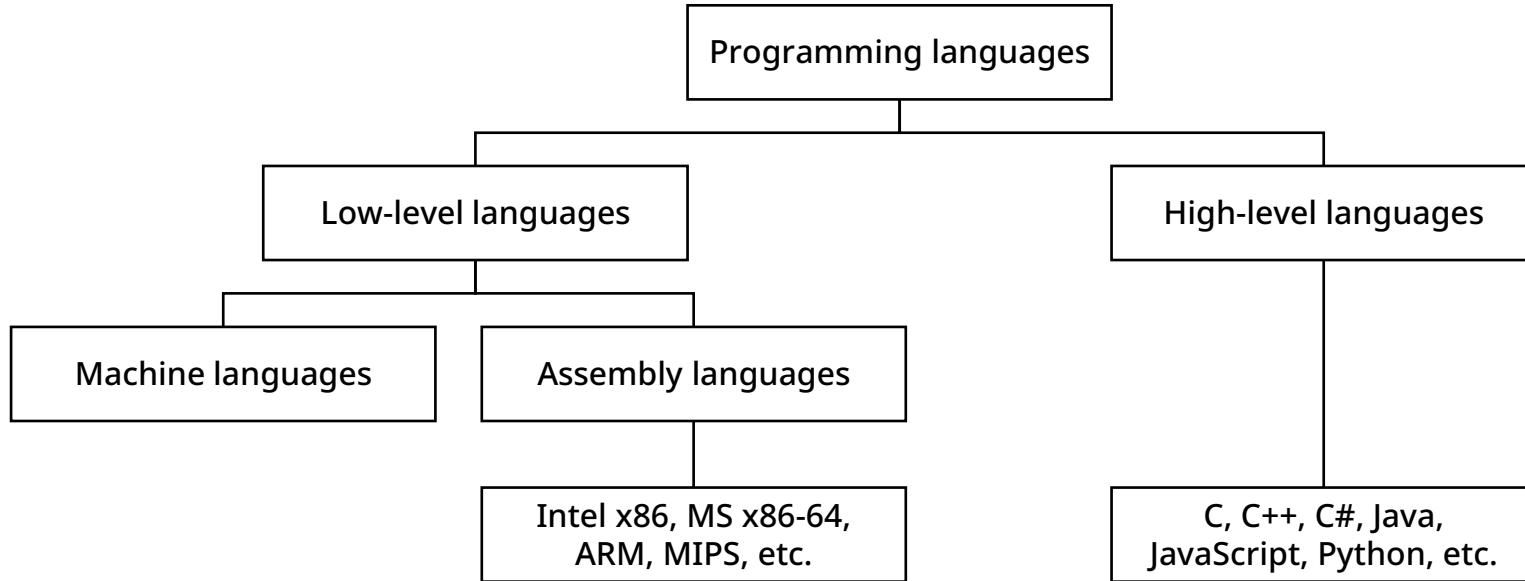
- The way a programming language categorizes values and expressions into types, which defines how those types are manipulated and how they interact with each other.

Understanding the programming language

Classifying programming languages

A programming language is a formal tool for writing software to run on computer systems. The more advanced the language, the more closely it resembles human language.

- Classification of programming languages



Understanding the programming language

Classifying programming languages (1) - processing levels

A programming language is a formal tool for writing software to run on computer systems. The more advanced the language, the more closely it resembles human language.

- Classification by processing level
 - Low-level programming languages
 - Easy for computers to understand by design
 - Fast processing speed
 - Poor portability and readability
 - High-level programming languages
 - Easy for humans to understand
 - Slow processing
 - High portability and readability

Understanding the programming language

Classifying programming languages (1) - processing levels

A programming language is a formal tool for writing software to run on computer systems. The more advanced the language, the more closely it resembles human language.

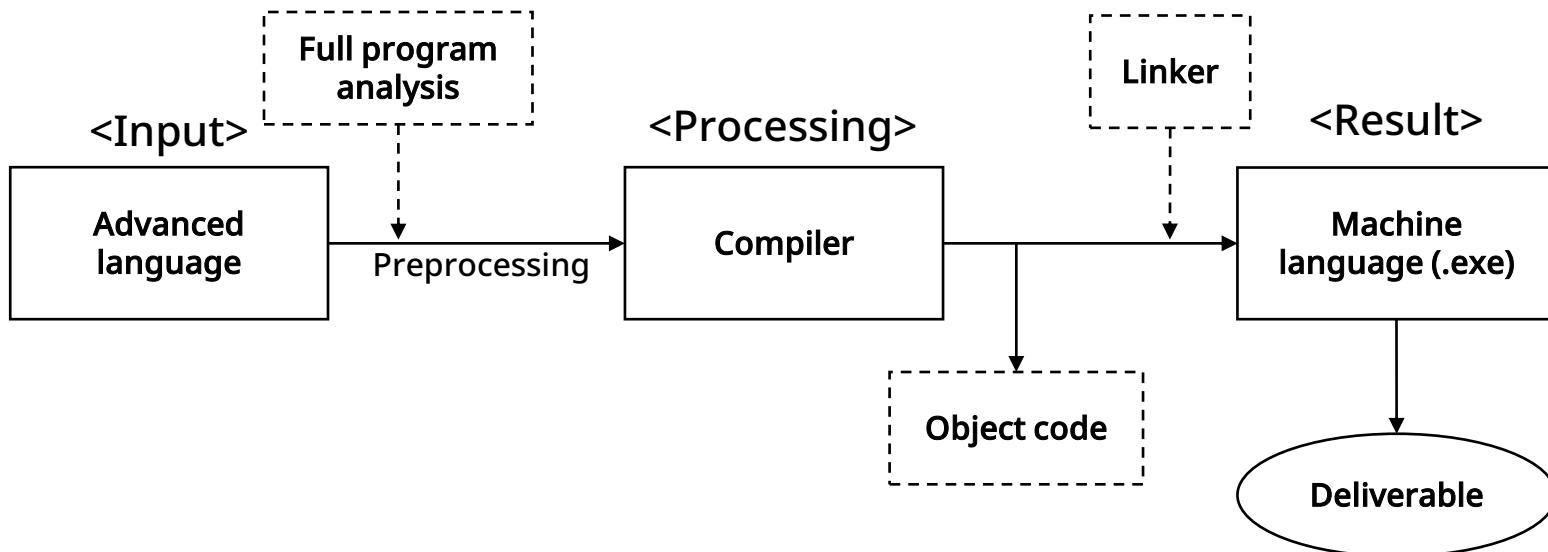
- Classification by processing level
 - Types of high-level languages
 - Compiled languages
 - C, C++, Go, Rust, etc.
 - Interpreted languages
 - PHP, HTML, JavaScript, etc.
 - Just-In-Time (JIT) compiled languages
 - C#, Java, Python, etc.

Understanding the programming language

Classifying programming languages (2) – ways of interpretation

A programming language is a formal tool for writing software to run on computer systems. The more advanced the language, the more closely it resembles human language.

- Compiler vs. interpreter vs. JIT
 - Process with a compiler
 - Compile to machine language by compiling all of the source code before running the program at compile time

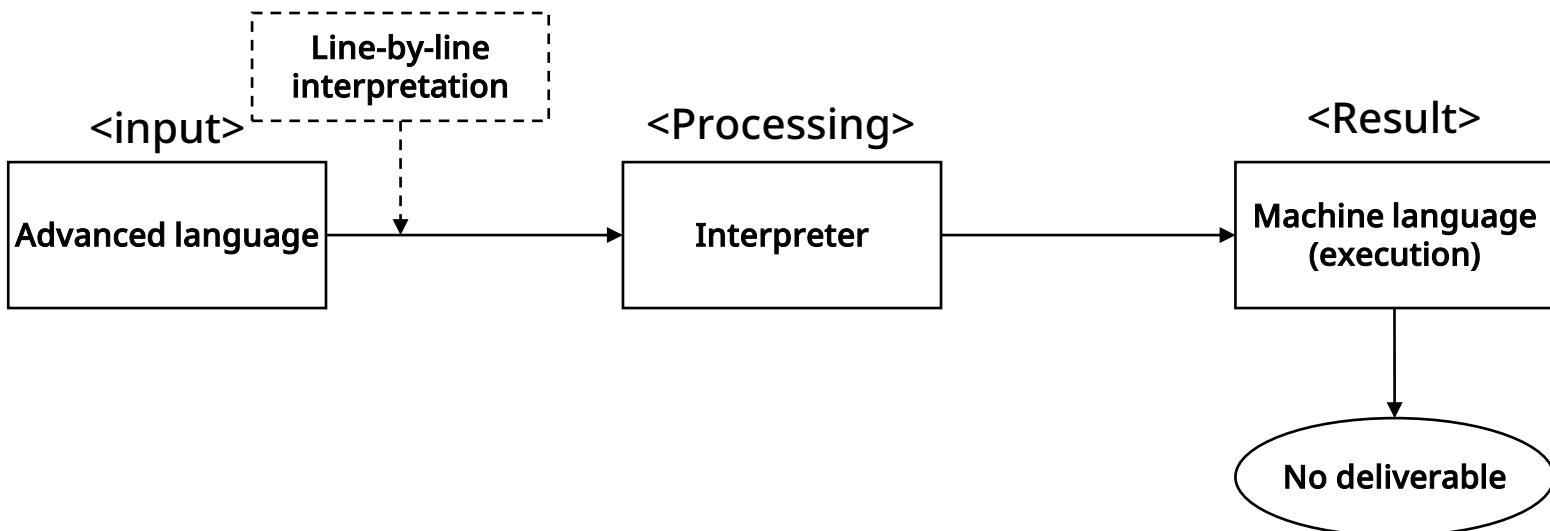


Understanding the programming language

Classifying programming languages (2) – ways of interpretation

A programming language is a formal tool for writing software to run on computer systems. The more advanced the language, the more closely it resembles human language.

- Compiler vs. interpreter vs. JIT
 - Process with an interpreter
 - Compile a source code line by line as a program runs at compile time



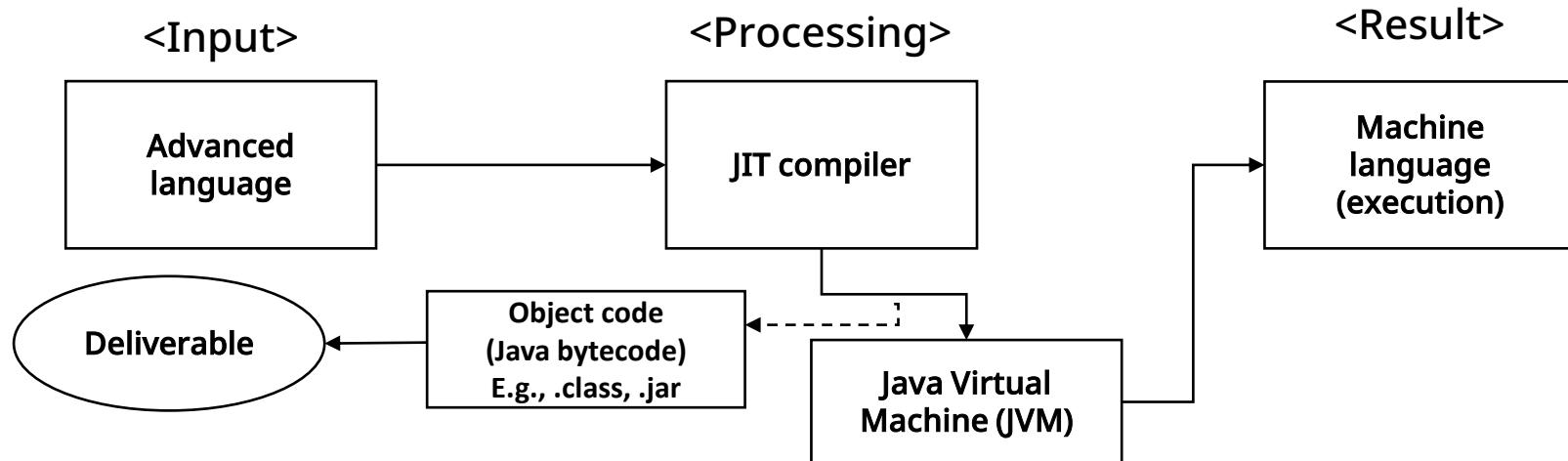
Understanding the programming language

Classifying programming languages (2) – ways of interpretation

A programming language is a formal tool for writing software to run on computer systems. The more advanced the language, the more closely it resembles human language.

- Compiler vs. interpreter vs. JIT

- Process with a JIT compiler
 - Compile all of the source code into a specific code before running the program
 - Run with the specific code compiled on each subsequent run after that



Understanding the programming language

Classifying programming languages (2) – ways of interpretation

A programming language is a formal tool for writing software to run on computer systems. The more advanced the language, the more closely it resembles human language.

- Compiler vs. interpreter vs. JIT

	Compiler	Interpreter	JIT compiler
Input	All Programs	Single-line code or command	All Programs
Deliverable	Generate intermediate object code (later deprecated)	X	Generating intermediate object code (later used)
Mechanism	Compile before running	Compile and execute at the same time	Compile and execute
Speed	Fast	Slow	Average
Memory	Much required	Less required	Required on average
Error detection	Hard	Easy	Average
Programming language	C, C++, Scala, Rust, etc.	JavaScript, Perl, PHP, etc.	Java, Python, etc.

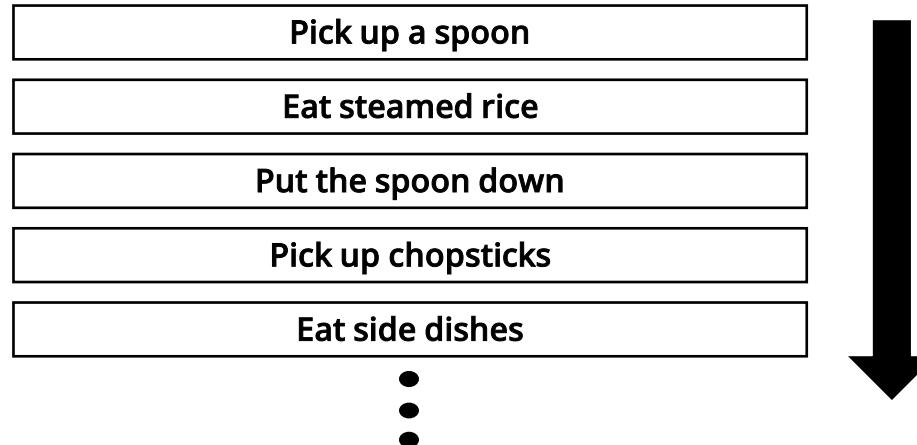
Understanding the programming language

Classifying programming languages (3) - paradigms

Programming involves creating a program by organizing and arranging formulas and operations to fit a computer, and then transforming them into computer-specific command code.

- Procedural languages vs. object-oriented languages
 - Procedural programming
 - Programming paradigm that uses routines, subroutines, methods, functions, etc., rather than simply executing sequential commands.

<Procedural language>



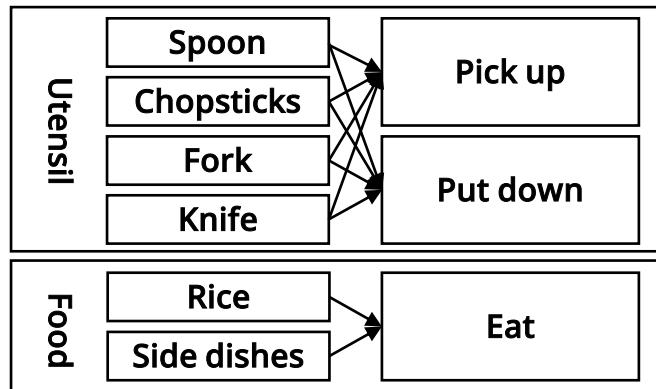
Understanding the programming language

Classifying programming languages (3) - paradigms

Programming involves creating a program by organizing and arranging formulas and operations to fit a computer, and then transforming them into computer-specific command code.

- Procedural languages vs. object-oriented languages
 - Object-oriented programming
 - Programming paradigm that executes commands on an object-by-object basis

<Object-oriented language>

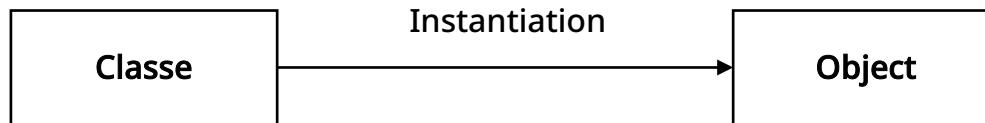


Understanding the programming language

Classifying programming languages (3) - paradigms

Programming involves creating a program by organizing and arranging formulas and operations to fit a computer, and then transforming them into computer-specific command code.

- Procedural languages vs. object-oriented languages
 - Object-oriented programming
 - Object : a thing or concept that exists in the real world.
 - Classe : a blueprint for creating objects
 - Instance : an actual occurrence of an object based on a class



Understanding the programming language

Classifying programming languages (3) - paradigms

Programming involves creating a program by organizing and arranging formulas and operations to fit a computer, and then transforming them into computer-specific command code.

- Procedural languages vs. object-oriented languages
 - Differences between procedural and object-oriented languages

Procedural language	Object-oriented language
The program is divided into procedural units.	The program is devided into objects.
Data and functions are separated.	Data and methods are unified.
Each unit is executed in sequential processes.	Commands are executed in non-sequential processes.
Not easy to improve development productivity, maintenance and scaling	Easy to improve development productivity, maintenance, and scaling
Fast to run	Slow to run

Understanding the programming language

Programming language rankings

- As of November 2023

- 1st : Python
- 2nd : C
- 3rd : C++

Nov 2023	Nov 2022	Change	Programming Language	Ratings	Change
1	1		 Python	14.16%	-3.02%
2	2		 C	11.77%	-3.31%
3	4	▲	 C++	10.36%	-0.39%
4	3	▼	 Java	8.35%	-3.63%
5	5		 C#	7.65%	+3.40%
6	7	▲	 JavaScript	3.21%	+0.47%
7	10	▲	 PHP	2.30%	+0.61%
8	6	▼	 Visual Basic	2.10%	-2.01%
9	9		 SQL	1.88%	+0.07%
10	8	▼	 Assembly language	1.35%	-0.83%

Source: <https://www.tiobe.com/tiobe-index/>

Understanding the programming language

Online programming sites

- <https://www.codecademy.com/>

The screenshot shows the homepage of Codecademy. On the left sidebar, there are several categories: Cybersecurity, For Business, Beta Catalog, Math, Open Source, Cloud Computing, DevOps, Data Engineering, Data Analytics, IT, Full Catalog, and Where do I begin?. The main content area is titled "Most popular courses". It features six course cards arranged in two rows of three. The first row includes "Learn Python 3" (Course, With Certificate, Beginner Friendly, 25 hours), "Learn HTML" (Free course, Beginner Friendly, 7 hours), and "Learn JavaScript" (Free course, Beginner Friendly, 15 hours). The second row includes "Learn Java" (Free course), "Learn SQL" (Free course), and "Learn C++" (Free course).

Category	Course Title	Type	Details
Most popular courses	Learn Python 3	Course	With Certificate, Beginner Friendly, 25 hours
	Learn HTML	Free course	Beginner Friendly, 7 hours
	Learn JavaScript	Free course	Beginner Friendly, 15 hours
Where do I begin?	Learn Java	Free course	
	Learn SQL	Free course	
	Learn C++	Free course	

Understanding the programming language

Online programming sites

- <https://www.hackerrank.com/>

Certification

Problem Solving (Basic) ⓘ

Get Certified



Python (Basic) ⓘ

Get Certified



Stand out from the crowd

Take the HackerRank Certification Test and make your profile stand out

[View all certifications](#)

Prepare By Topics

 Algorithms	 Data Structures	 Mathematics
 Artificial Intelligence	 C	 C++
 Java	 Python	 Ruby
 SQL	 Databases	 Linux Shell
 Functional Programming	 Regex	

Preparation Kits [View all kits](#)

1 Week Preparation Kit

Challenges: 21, Attempts: 780879, Mock Tests: 6

Problem Solving (Basic) Problem Solving (Intermediate) +1



1 Month Preparation Kit

Challenges: 54, Attempts: 362903, Mock Tests: 4

Problem Solving (Basic) Problem Solving (Intermediate) +1



02

C programming



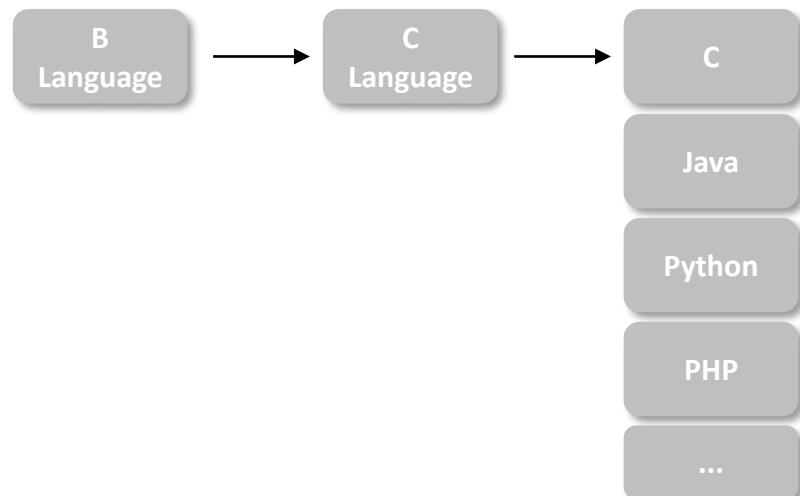
- C Programming Overview
- C Programming Environment Setup
- Structure of the C Program
- C - Variables and Data Types
- C - Operator
- C - Standard Data Input/Output
- C - Control Statements
- C - Functions
- C - Array and Pointer
- C - Structure
- C - Strings
- C - File Input/Output
- C - Memory Management
- C - Preprocessors
- C - Data Structure

C programming overview

C language overview

C is a programming language developed in 1972 by Ken Thompson and Dennis Ritchie for use on the newly developed Unix operating system while working at Bell Labs.

- How the C language came to be
 - Created to address the development challenges of Unix, the once-dominant operating system
 - The problem of having to rewrite to assembly every time the CPU structure changed
 - Increased need for high-level languages that any developer could understand and modify for programming



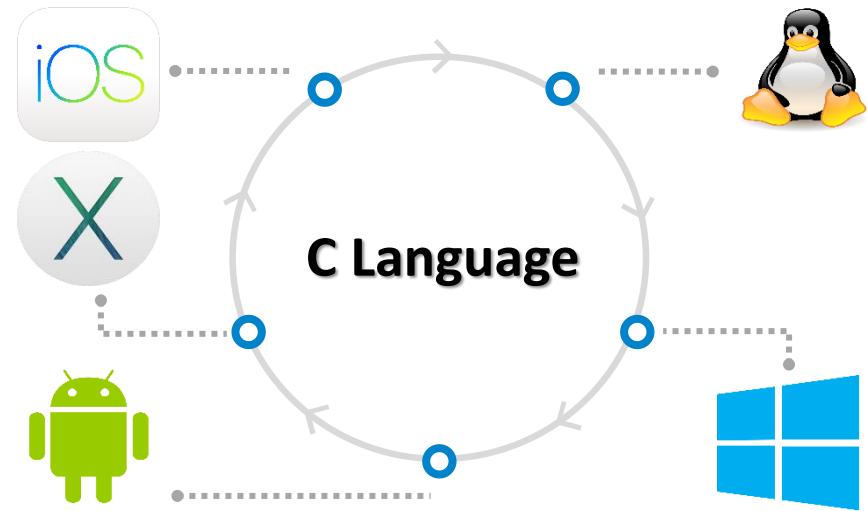
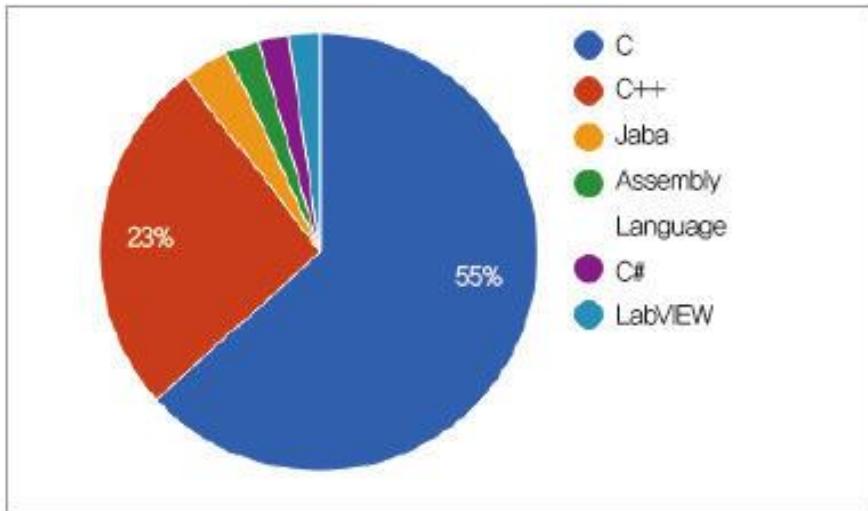
Sources : Wikipedia, <https://www.tcpschool.com/>

C programming overview

C Language overview

C is a programming language developed in 1972 by Ken Thompson and Dennis Ritchie for use on the newly developed Unix operating system while working at Bell Labs.

- The C language since its birth
 - In the recent IoT era, it has re-emerged as a key language due to the growing demand for embedded development.
 - The kernel, the heart of many operating systems including Windows, Linux, OS X & iOS, and Android, is also developed in C.



Sources : Embedded Dragon Consulting & VDC Research

C programming overview

C Language overview

As with any language, C has its advantages and disadvantages. To understand them, you need to comprehend the main purpose and philosophy of the language. The main purpose is undoubtedly to create operating systems or system software.

- Features of the C language

Hardware control

- Control hardware at the assembly language level
- Have low-level language features that are more difficult to learn than other more advanced languages

Scalability

- Standardized by ANSI and ISO
- Help programs compile and run without CPU or OS involvement

Fast runtime

- Have a simple program structure as a procedural language
- Low program complexity for efficient system programming

Flexibility

- Can be used in many areas such as factory automation, office automation, GUIs, applications
- Allow most features not allowed in other languages

Standard library

- It is a repository of hundreds of useful functions for input/output, string processing, memory allocation, and more.

C programming environment setup

Installing the IDE

The IDE is a source code editor developed by Microsoft Corporation for Microsoft Windows, macOS, and Linux. It includes debugging support, Git control, syntax highlighting, and allows users to change the editor's theme, shortcuts, settings, and more.

- Lab environments
 - Operating System : Windows 10
 - Programs used : Visual Studio Code 1.85 (latest), MinGW 2013-10-26 (latest)

C programming environment setup

Installing the IDE

The IDE is a source code editor developed by Microsoft Corporation for Microsoft Windows, macOS, and Linux. It includes debugging support, Git control, syntax highlighting, and allows users to change the editor's theme, shortcuts, settings, and more.

- Visual Studio (VS) Code overview
 - First introduced at the Microsoft Build conference on April 29, 2015, when a preview version was released to the public.
 - Released under the MIT License on November 18, 2015
 - Source code available on GitHub and extension support also announced
 - The public preview phase ended on April 14, 2016, and the full version was released over the web.
 - VS Code features
 - As a source code editor, it supports many programming languages.
 - Provide handy features that can be used with each language
 - Add new extensions through plugins, such as additional editing features and programming language support

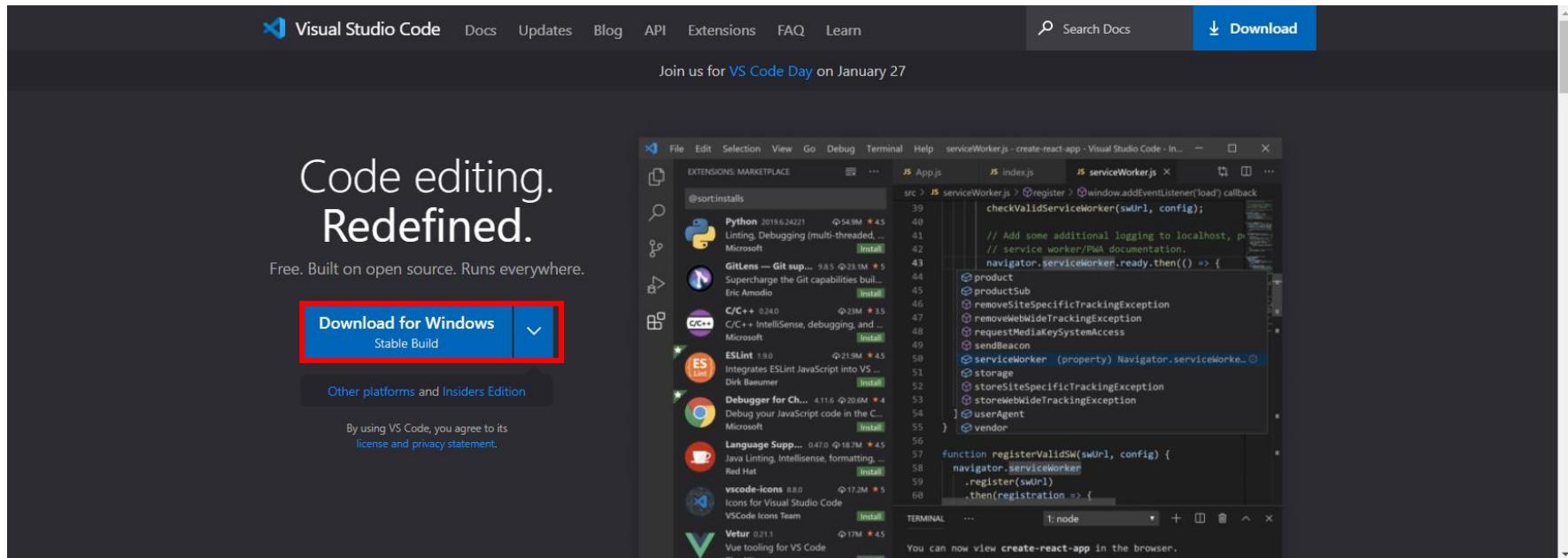
C programming environment setup

Installing the IDE

The IDE is a source code editor developed by Microsoft Corporation for Microsoft Windows, macOS, and Linux. It includes debugging support, Git control, syntax highlighting, and allows users to change the editor's theme, shortcuts, settings, and more.

- How to install VS Code

- Go to <https://code.visualstudio.com/> and click on the appropriate button.

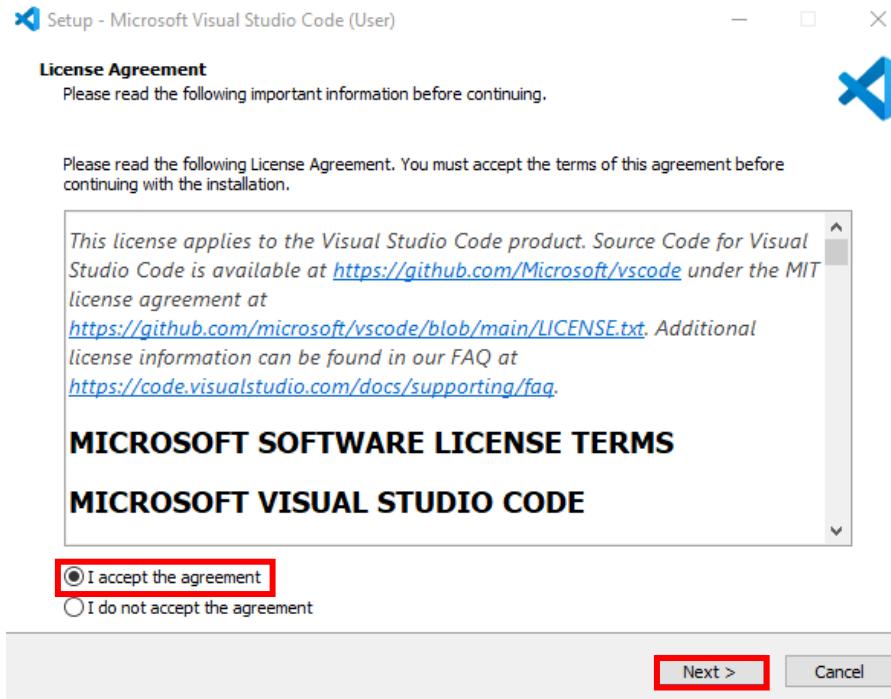


C programming environment setup

Installing the IDE

The IDE is a source code editor developed by Microsoft Corporation for Microsoft Windows, macOS, and Linux. It includes debugging support, Git control, syntax highlighting, and allows users to change the editor's theme, shortcuts, settings, and more.

- How to install VS Code
 - Click on the downloaded .exe file, accept the License Agreement, and the “Next>” button.

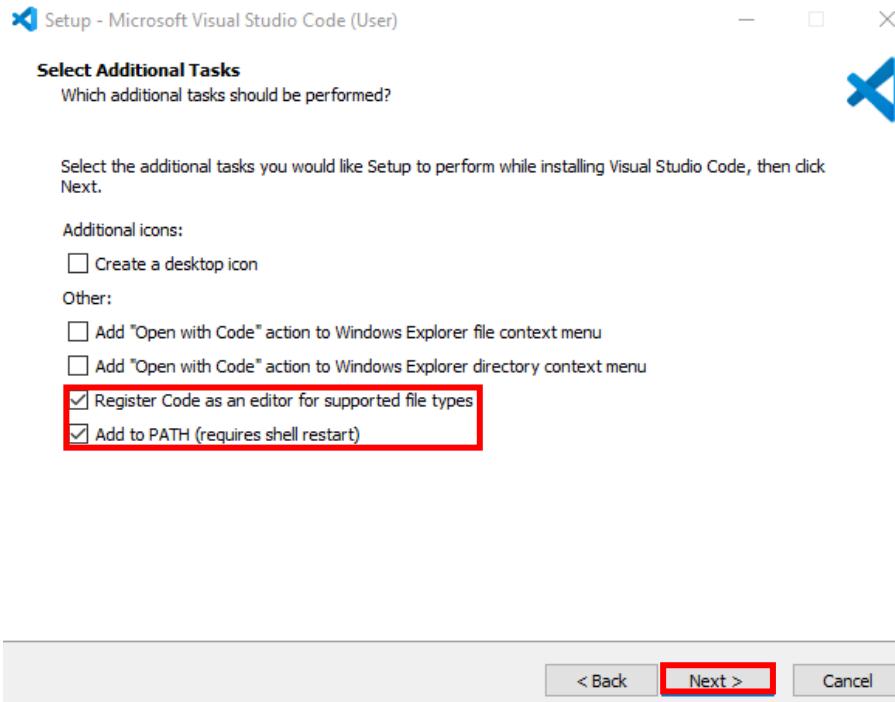


C programming environment setup

Installing the IDE

The IDE is a source code editor developed by Microsoft Corporation for Microsoft Windows, macOS, and Linux. It includes debugging support, Git control, syntax highlighting, and allows users to change the editor's theme, shortcuts, settings, and more.

- How to install VS Code
 - Confirm that you want to select the Start Menu folder, and then click on the “Next>” button.

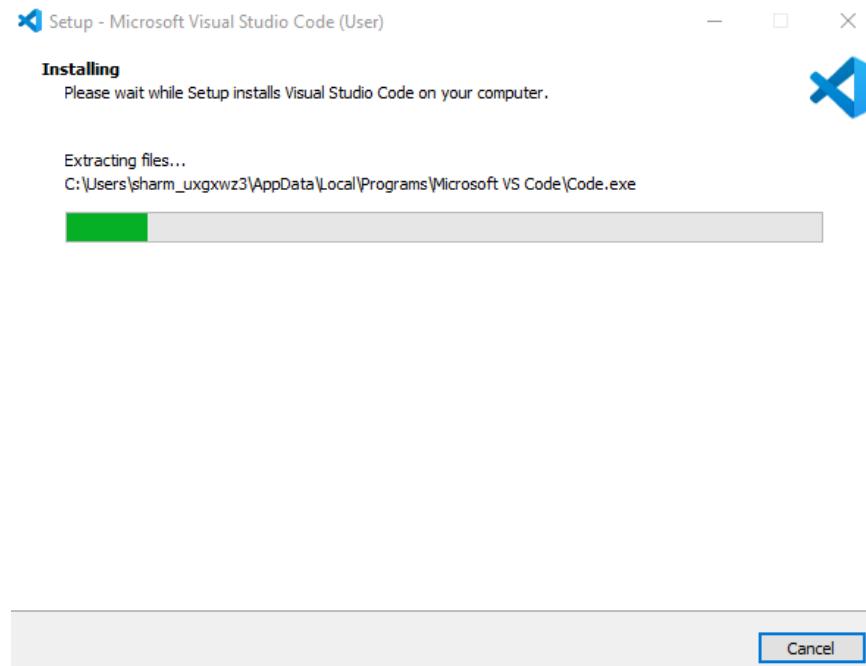
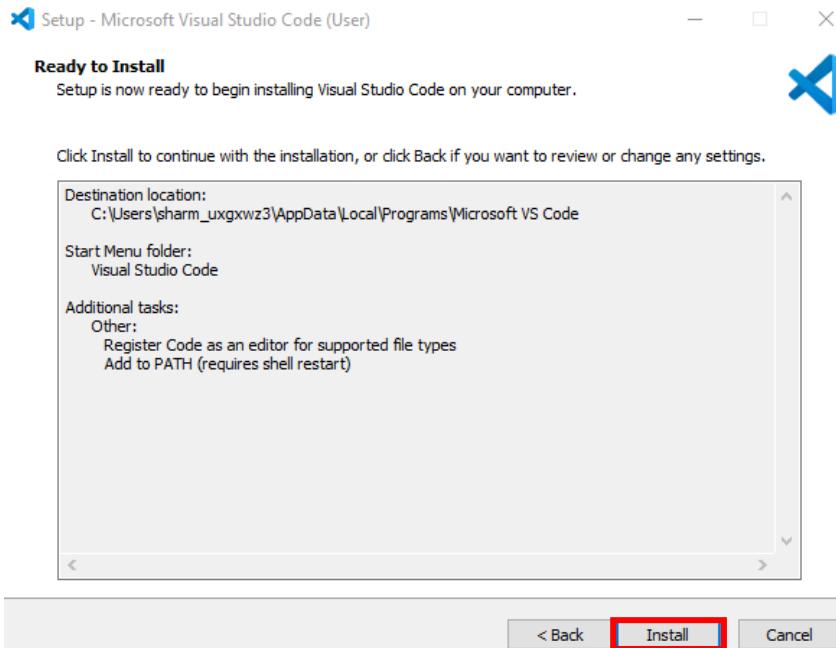


C programming environment setup

Installing the IDE

The IDE is a source code editor developed by Microsoft Corporation for Microsoft Windows, macOS, and Linux. It includes debugging support, Git control, syntax highlighting, and allows users to change the editor's theme, shortcuts, settings, and more.

- How to install VS Code
 - Confirm “Ready to Install” and click “Install.”
 - Proceed with the installation.



C programming environment setup

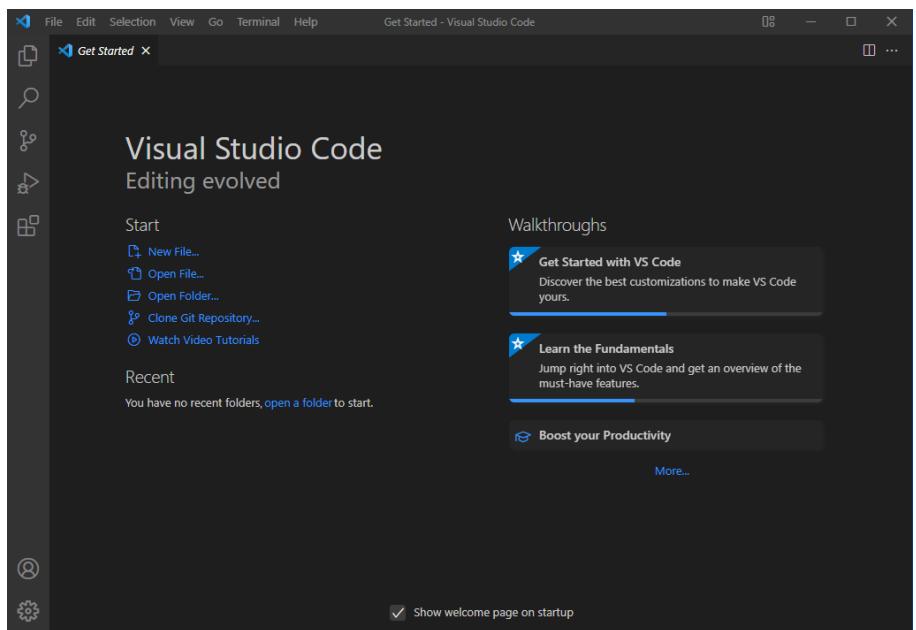
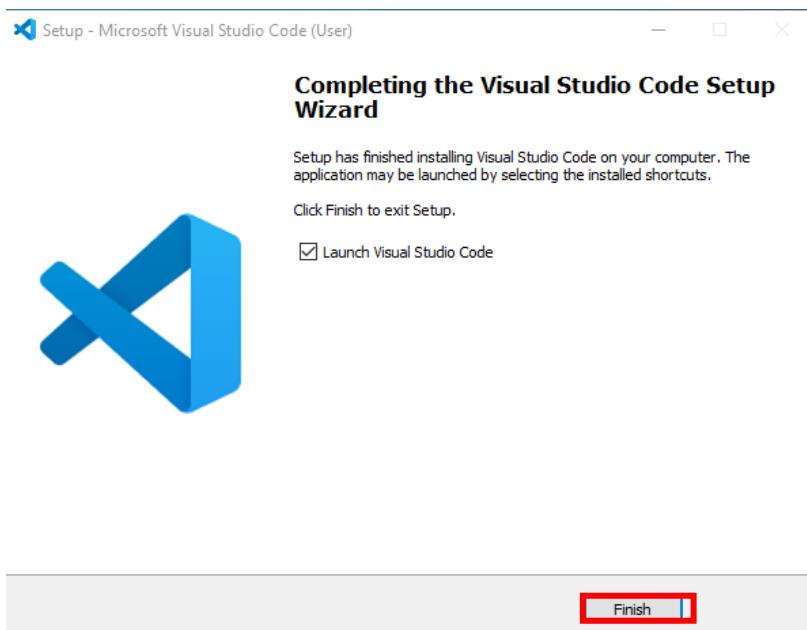
Installing the IDE

The IDE is a source code editor developed by Microsoft Corporation for Microsoft Windows, macOS, and Linux. It includes debugging support, Git control, syntax highlighting, and allows users to change the editor's theme, shortcuts, settings, and more.

- How to install VS Code

- Click "Finish" when the installation is complete.

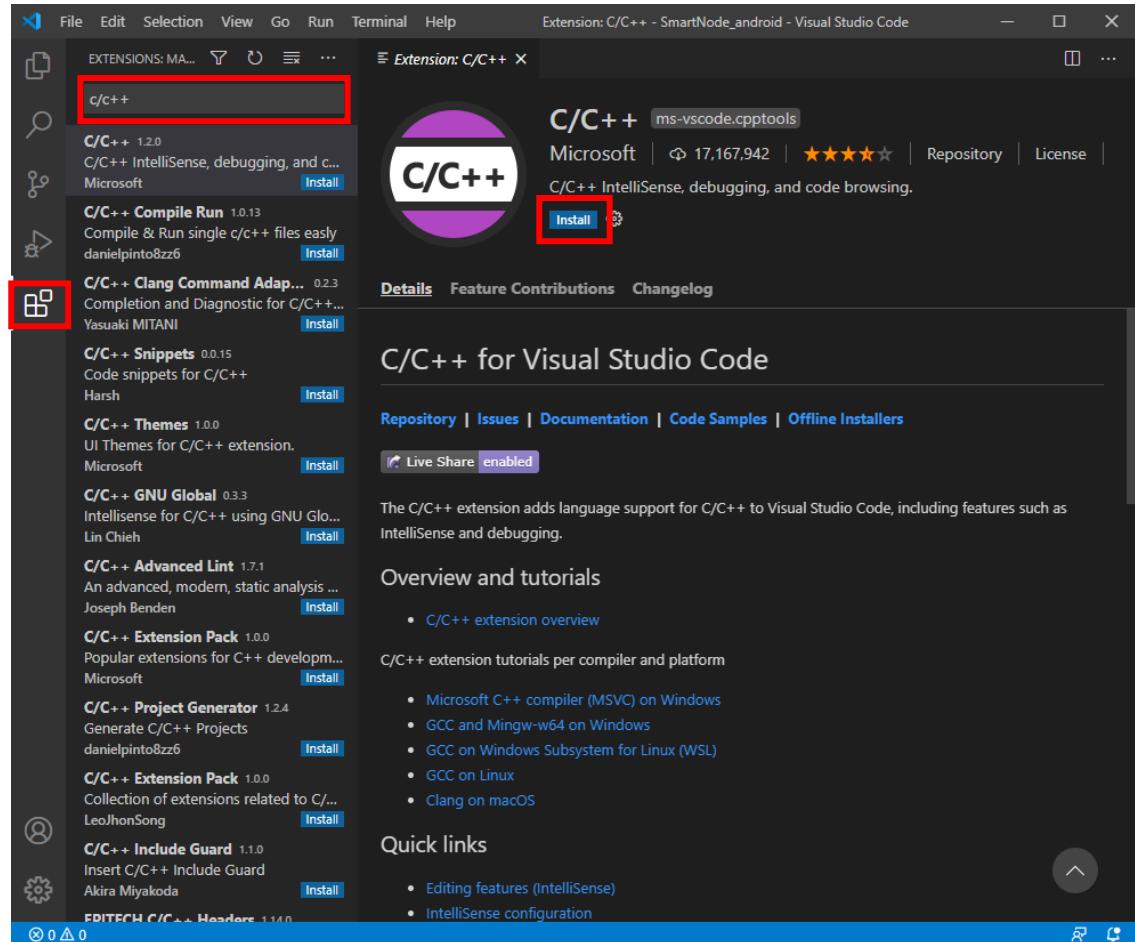
- Run screen after the IDE has finished installing.



C programming environment setup

Installing the IDE

- How to install VS Code GCC
 - Install the C/C++ extension
 - Click on the 5th tab on the left
 - Type C/C++ in the search box
 - Install the first extension



C programming environment setup

Installing the IDE

- How to install VS Code GCC
 - Download MinGW, the compiler
 - <https://sourceforge.net/projects/mingw/files/>

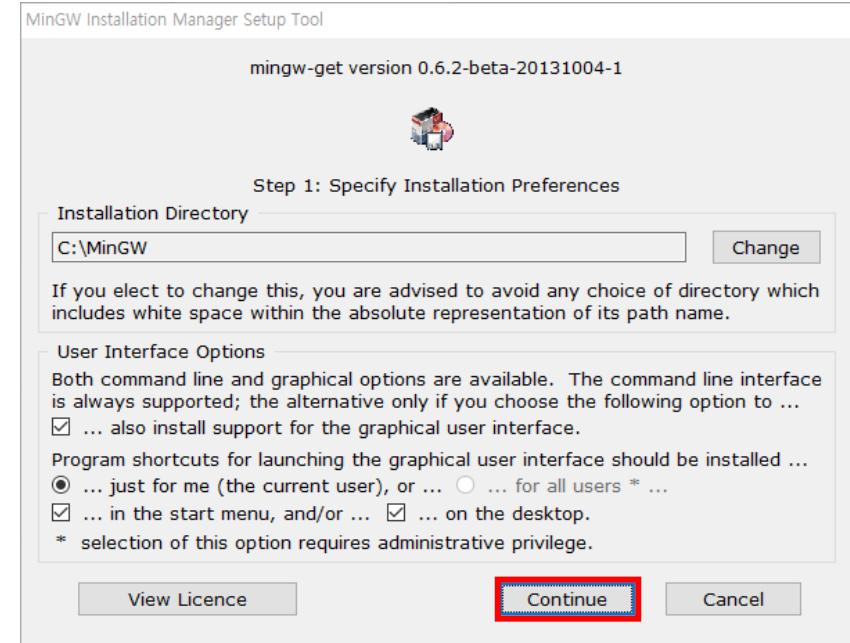
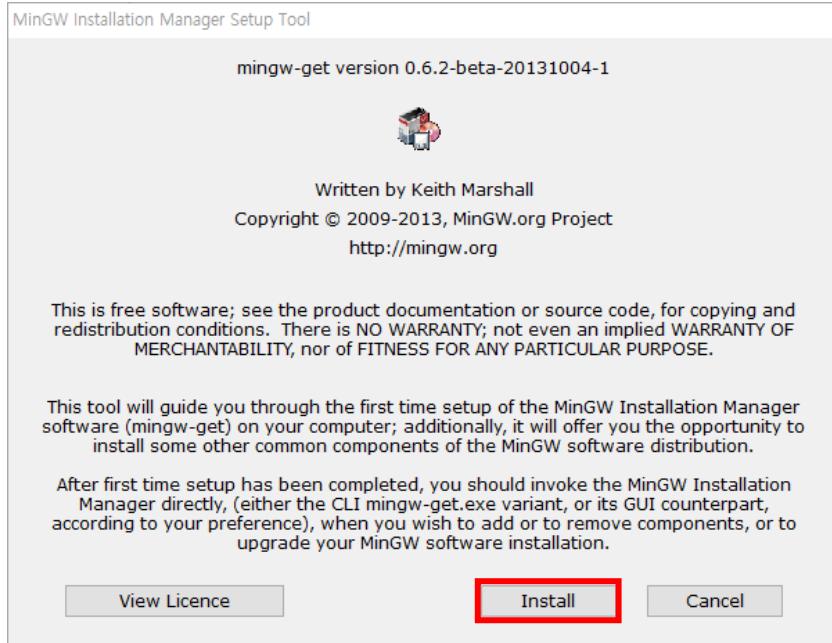
The screenshot shows the SourceForge website interface for the MinGW project. At the top, there's a navigation bar with links for Texas Instruments, a banner for 'A TOUGH DRIVER FOR A STRONGER SYSTEM', and advertisements for 'Gate Driver' and 'TI POWER'. Below the header, the main content area displays the project's homepage for 'MinGW - Minimalist GNU for Windows'. It features a logo, a brief description ('A native Windows port of the GNU Compiler Collection (GCC)'), and credits to contributors ('Brought to you by: cstraus, earnie, gressett, keithmarshall'). A prominent red button at the top says 'Download Latest Version mingw-get-setup.exe (86.5 kB)'. Below this, there's a navigation menu with tabs for Summary, Files (which is selected), Reviews, Support, News, Wiki, Mailing Lists, Tickets, and Git. The main content area lists five items in a table: 'MinGW' (modified 2013-10-26, 737,935 downloads/week), 'Installer' (modified 2013-10-04, 914,216 downloads/week), 'Other' (modified 2011-11-13, 718 downloads/week), 'MSYS' (modified 2011-11-13, 846,189 downloads/week), and 'README' (modified 2011-11-13, 44 downloads/week). A note at the bottom states: 'Welcome to the MinGW project file distribution directories. This is the top level directory containing Installer, MinGW, MSYS and'.

C programming environment setup

Installing the IDE

- C programming environment setup
 - How to install VS Code GCC

- After running the downloaded .exe file, click "Install." - Continue the installation.

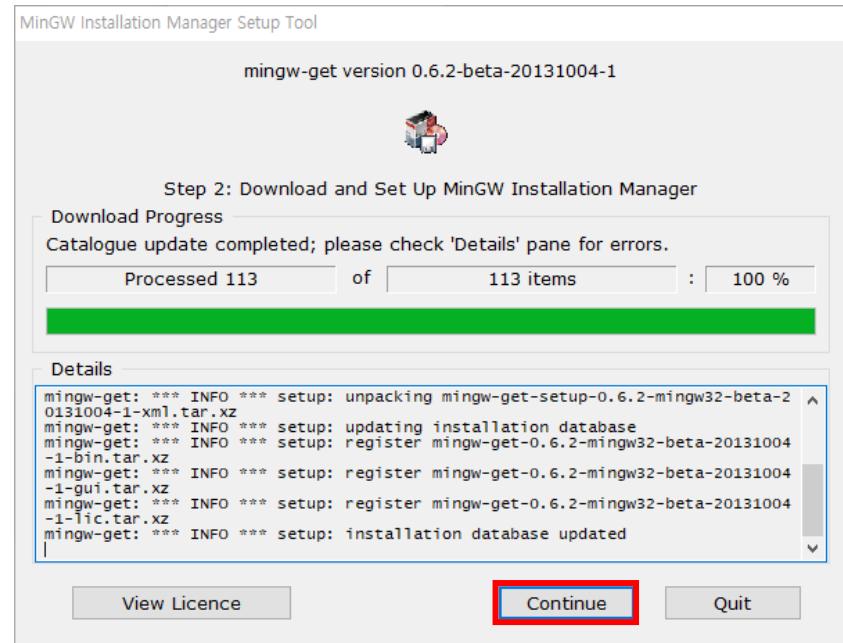
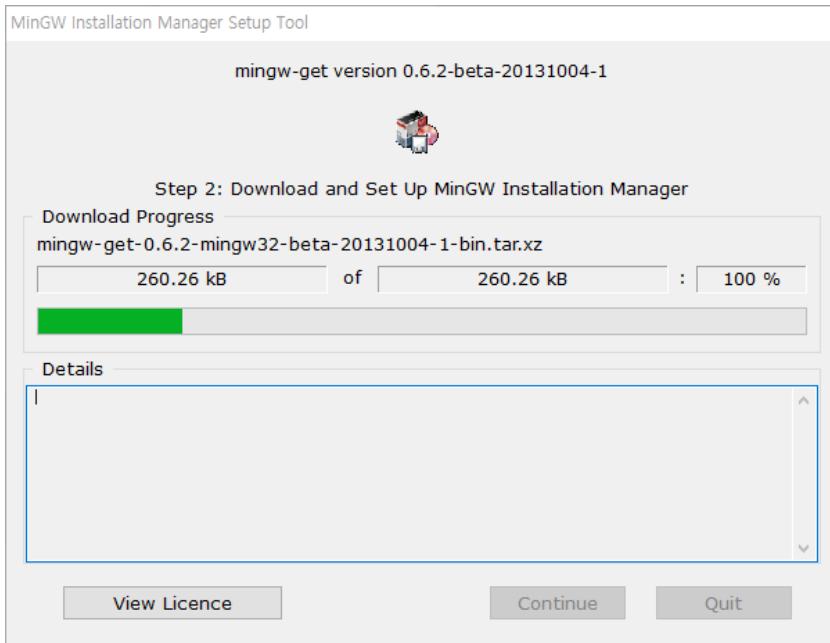


C programming environment setup

Installing the IDE

- C programming environment setup
 - How to install VS Code GCC
 - Proceed with the installation.

- Click "Continue" when the installation is complete.



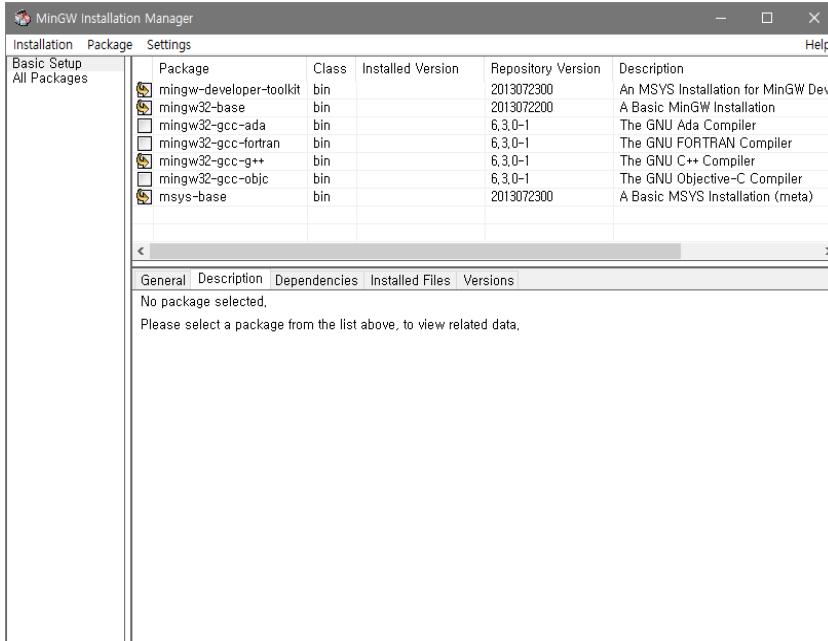
C programming environment setup

Installing the IDE

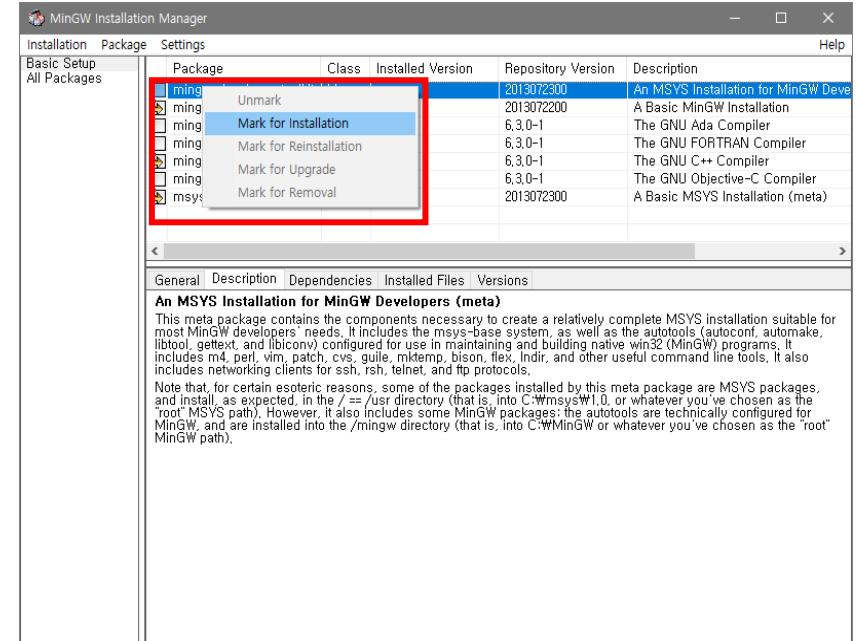
- C programming environment setup

- How to install VS Code GCC

- Check the MinGW Installation Manager screen.



- Right-click on packages to install and select "Mark for Installation".



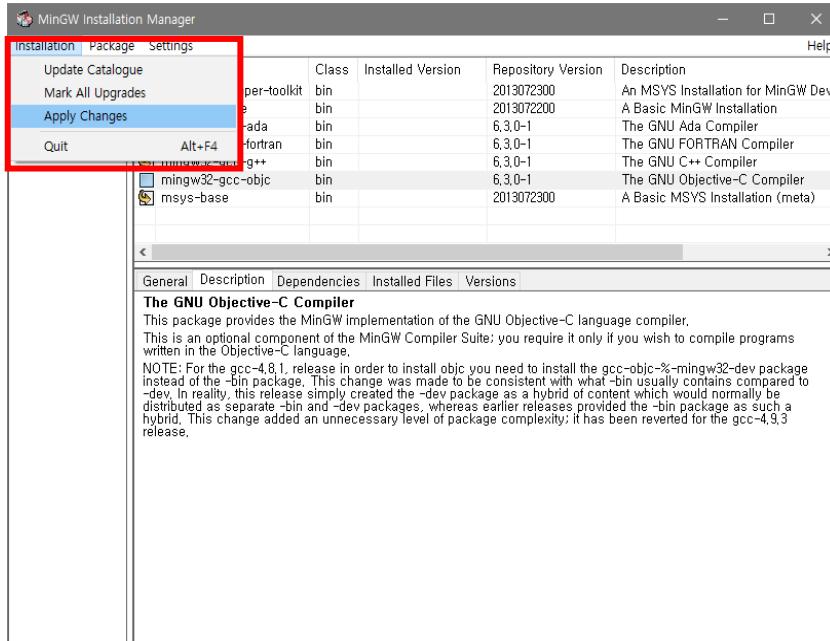
- mingw-developer-toolkit
- mingw32-base
- mingw32-gcc-g++
- msys-base

C programming environment setup

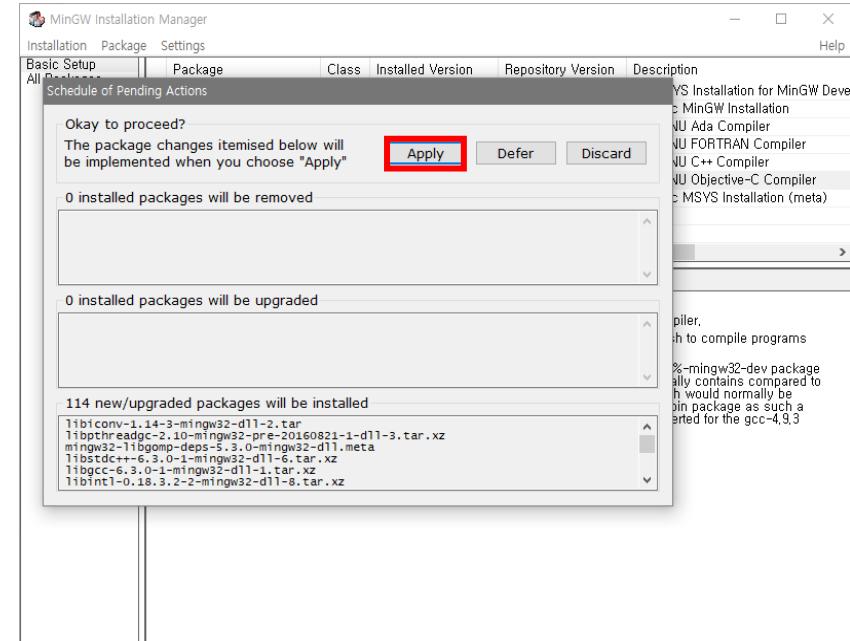
Installing the IDE

- C programming environment setup
 - How to install VS Code GCC

- On the title bar, “Installation” → “Apply Changes”,



- Click on the “Apply” button.



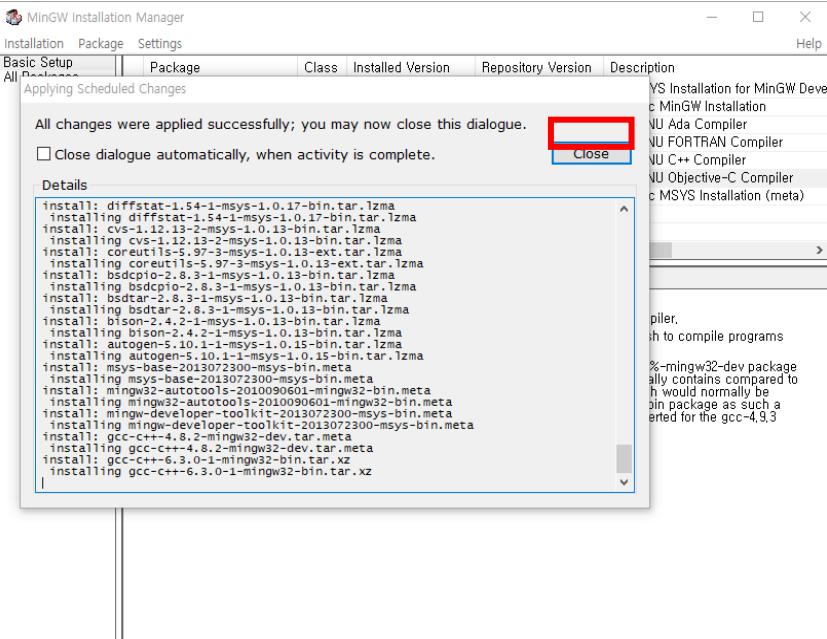
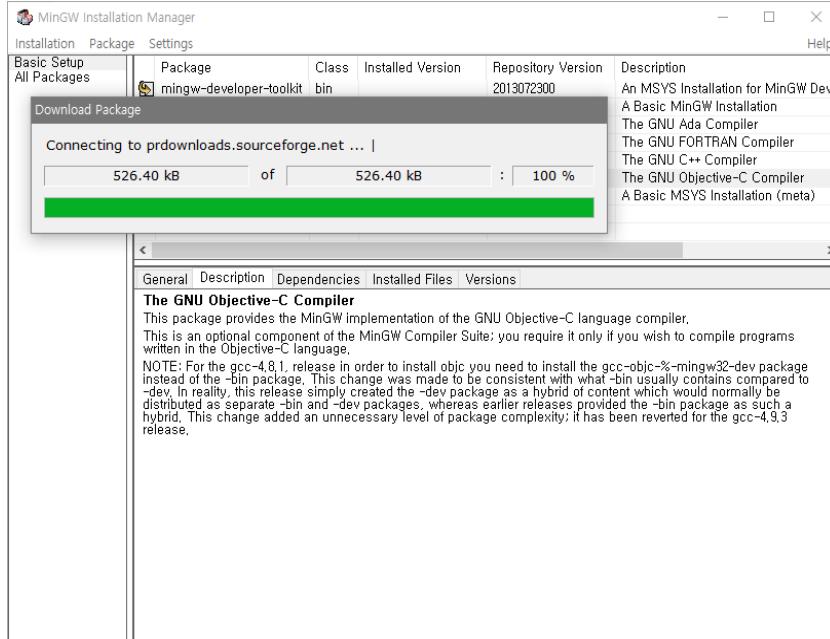
C programming environment setup

Installing the IDE

- C programming environment setup

- How to install VS Code GCC

- Proceed with the package installation.

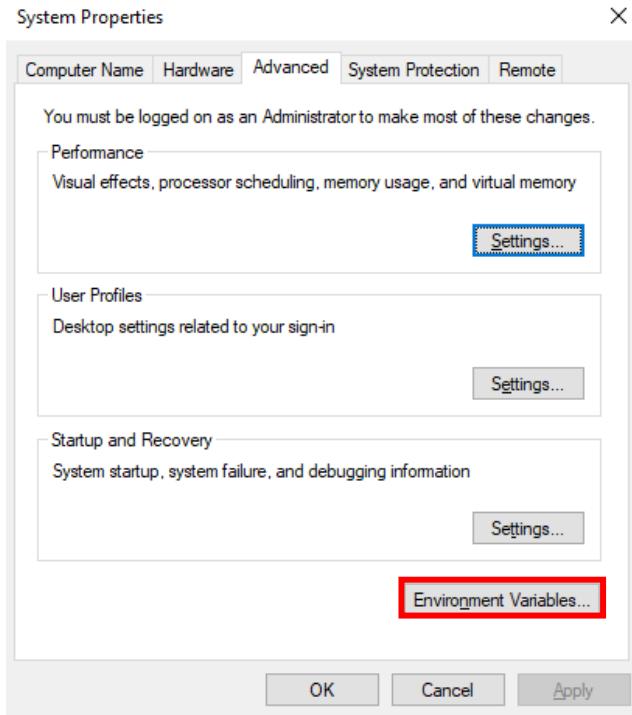


C programming environment setup

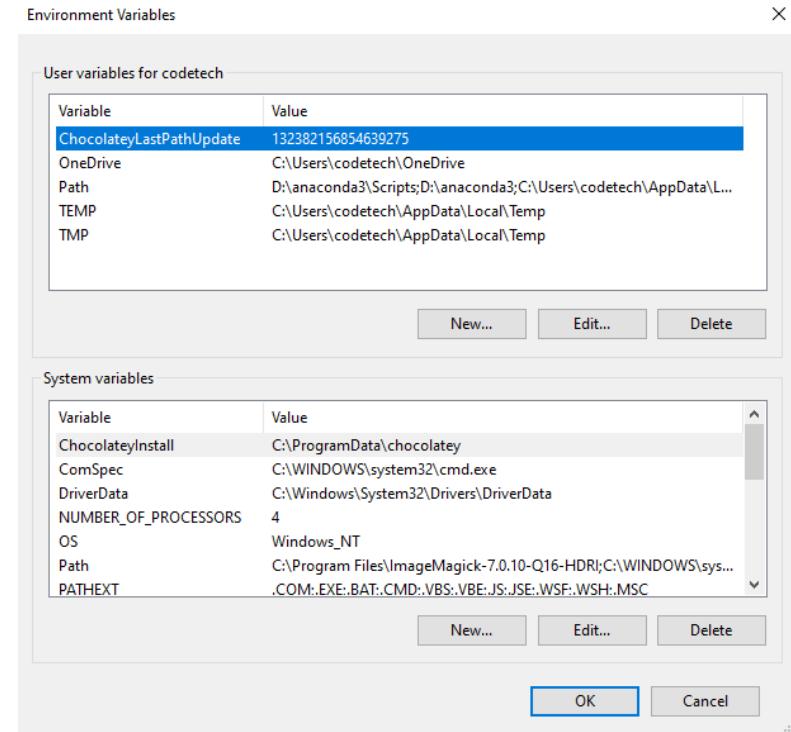
Installing the IDE

- C programming environment setup
 - How to install VS Code GCC

- Open “System Properties”.



- Open “Environment Variables”.



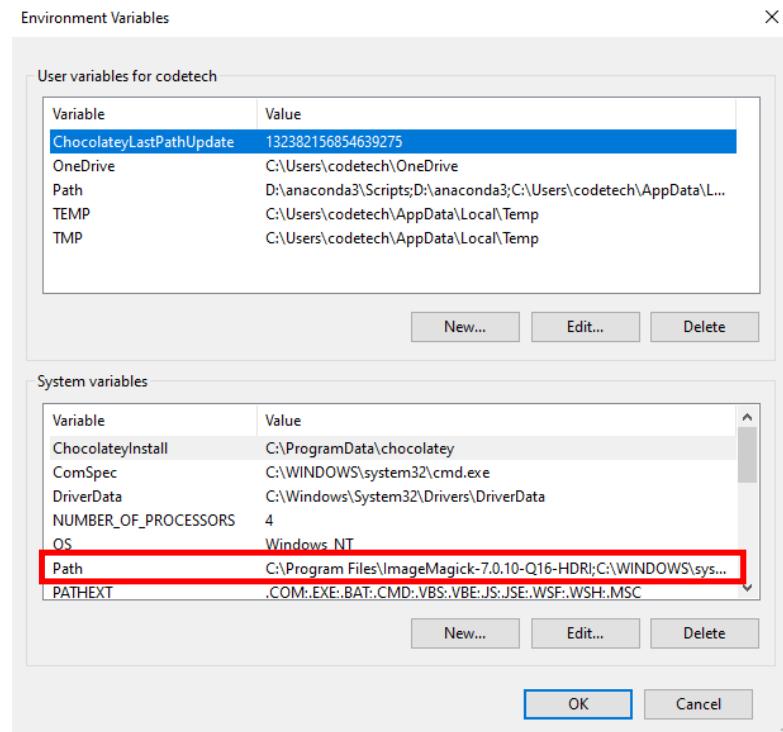
C programming environment setup

Installing the IDE

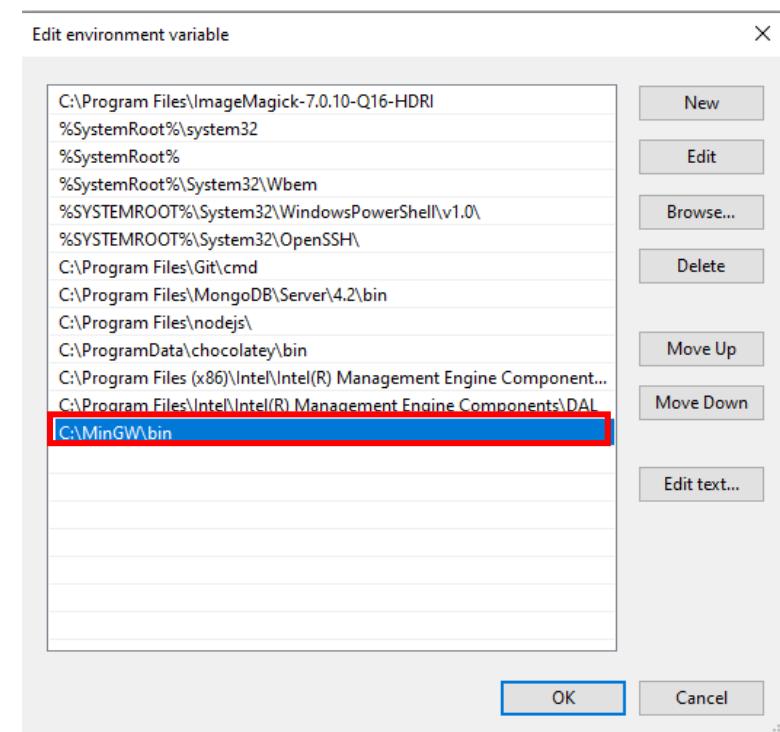
- C programming environment setup

- How to install VS Code GCC

- On the “System Variables” box, select “Path”.



- Add MinGW\bin.

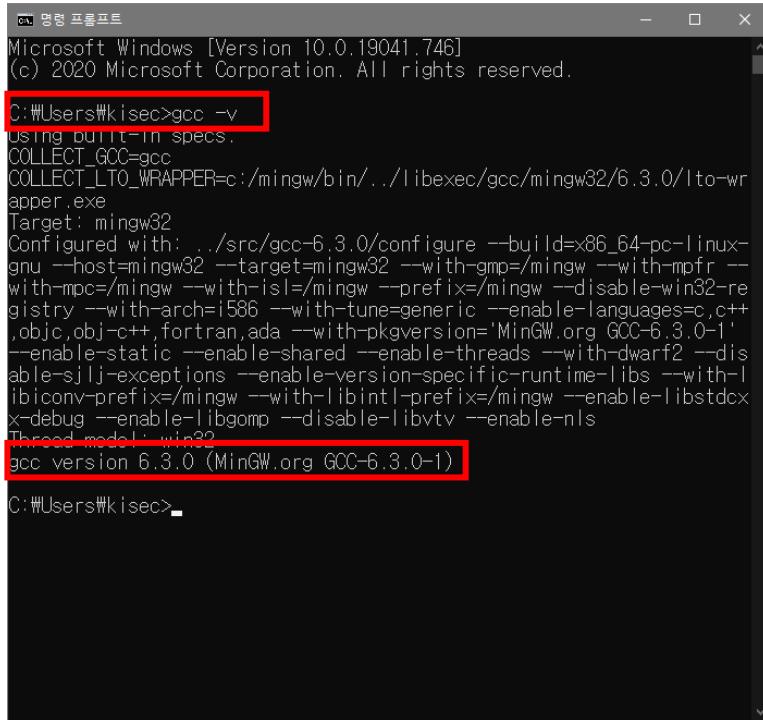


C programming environment setup

Installing the IDE

- C programming environment setup
 - How to install VS Code GCC

⑯ Check the GCC installation (gcc -v).

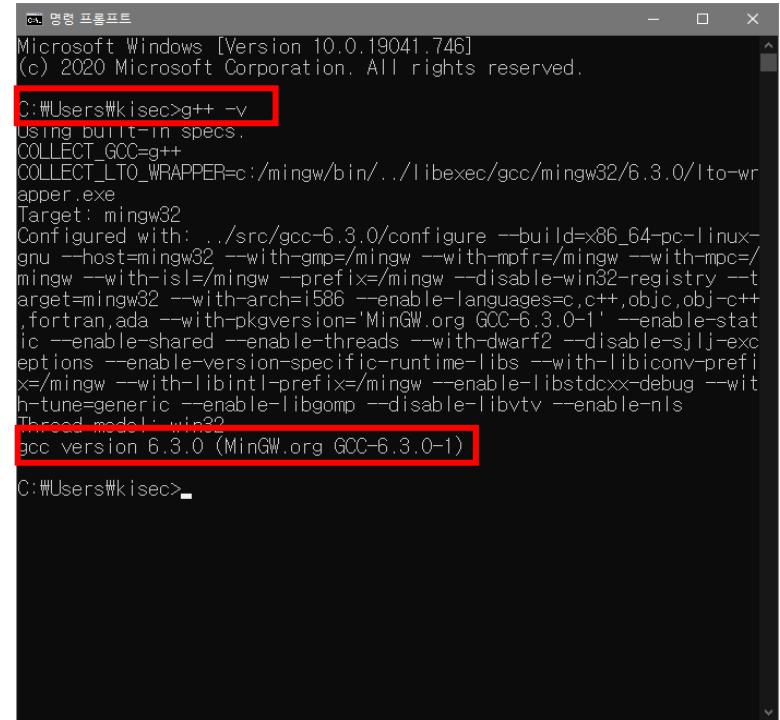


```
Microsoft Windows [Version 10.0.19041.746]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\#Users\kisec>gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=c:/mingw/bin/../../libexec/gcc/mingw32/6.3.0/lto-wr
apper.exe
Target: mingw32
Configured with: ../src/gcc-6.3.0/configure --build=x86_64-pc-linux-
gnu --host=mingw32 --target=mingw32 --with-gmp=/mingw --with-mpfr --
with-mpc=/mingw --with-isl=/mingw --prefix=/mingw --disable-win32-regis
try --with-arch=i586 --with-tune=generic --enable-languages=c,c+
,objc,obj-c++,fortran,ada --with-pkgversion='MinGW.org GCC-6.3.0-1'
--enable-static --enable-shared --enable-threads --with-dwarf2 --dis
able-sjlj-exceptions --enable-version-specific-runtime-libs --with-l
ibiconv-prefix=/mingw --with-libintl-prefix=/mingw --enable-libstdcxx-
x-debug --enable-libgomp --disable-libvtv --enable-nls
Thread model: win32
gcc version 6.3.0 (MinGW.org GCC-6.3.0-1)

C:\#Users\kisec>
```

⑰ Check the G++ installation (g++ -v).



```
Microsoft Windows [Version 10.0.19041.746]
(c) 2020 Microsoft Corporation. All rights reserved.

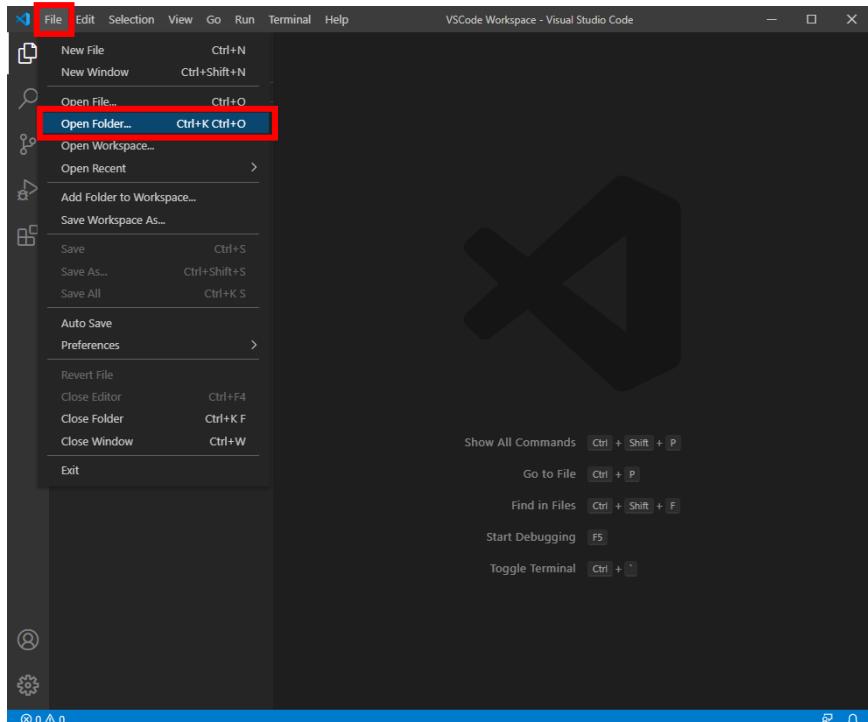
C:\#Users\kisec>g++ -v
Using built-in specs.
COLLECT_GCC=g++
COLLECT_LTO_WRAPPER=c:/mingw/bin/../../libexec/gcc/mingw32/6.3.0/lto-wr
apper.exe
Target: mingw32
Configured with: ../src/gcc-6.3.0/configure --build=x86_64-pc-linux-
gnu --host=mingw32 --with-gmp=/mingw --with-mpfr=/mingw --with-mpc=
mingw --with-isl=/mingw --prefix=/mingw --disable-win32-registry --
target=mingw32 --with-arch=i586 --enable-languages=c,c++,objc,obj-c+
,fortran,ada --with-pkgversion='MinGW.org GCC-6.3.0-1' --enable-stat
ic --enable-shared --enable-threads --with-dwarf2 --disable-sjlj-exc
ptions --enable-version-specific-runtime-libs --with-libiconv-prefix=
/mingw --with-libintl-prefix=/mingw --enable-libstdcxx-debug --wit
h-tune=generic --enable-libgomp --disable-libvtv --enable-nls
Thread model: win32
g++ version 6.3.0 (MinGW.org GCC-6.3.0-1)

C:\#Users\kisec>
```

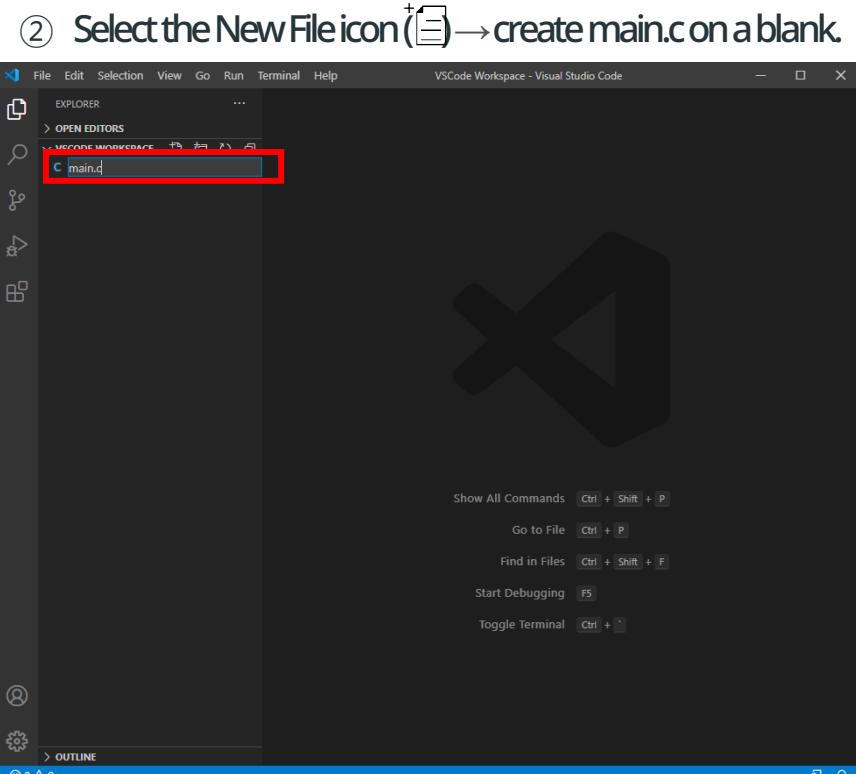
C programming environment setup

Installing the IDE

- C programming environment setup
 - How to set the VS Code GCC build
- ① “File” tab → “Open Folder” → select and open a folder.



- Create a folder to store code on any path.
 - E.g., D:\VSCode Workspace

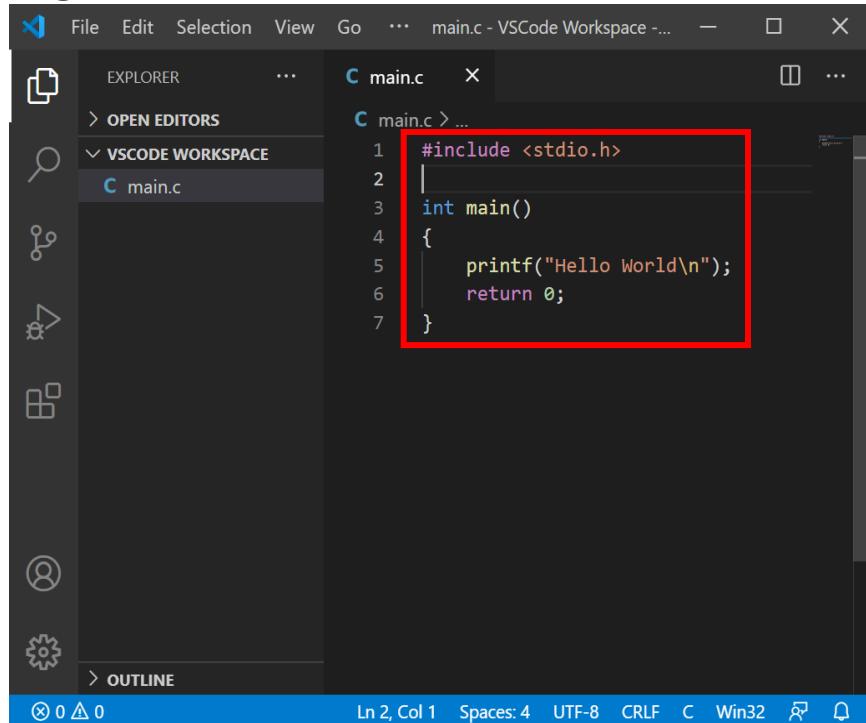


C programming environment setup

Installing the IDE

- C programming environment setup
 - How to set the VS Code GCC build

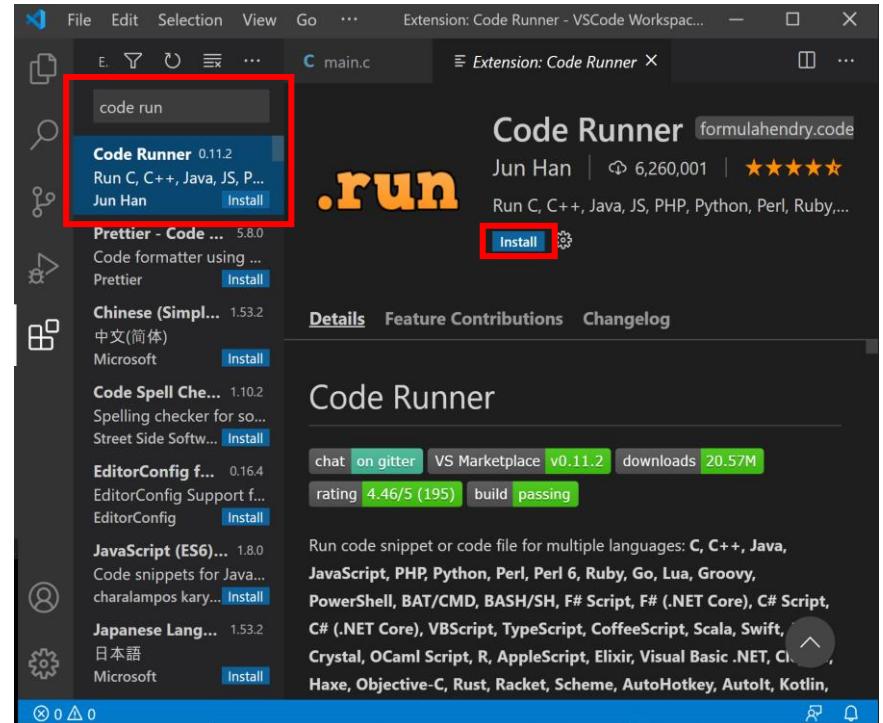
③ Write the main.c test code.



A screenshot of the Visual Studio Code interface. The title bar says "main.c - VSCode Workspace". The left sidebar shows the "EXPLORER" view with "VS CODE WORKSPACE" expanded, showing "main.c". The main editor area contains the following C code:

```
#include <stdio.h>
int main()
{
    printf("Hello World\n");
    return 0;
}
```

④ Install the extension to run a code.



- Click on the 5th icon on the left → search for code run.
- Select Code Runner → install.

C programming environment setup

Installing the IDE

- C programming environment setup

- How to set the VS Code GCC build
- Run the test code
 - Ctrl + Alt + N
 - Click the Play icon at the top right

The screenshot shows the Visual Studio Code interface with a dark theme. On the left is the Explorer sidebar with a 'VS CODE WORKSPACE' section containing 'main.c'. The main area shows a code editor with the following C code:

```
#include <stdio.h>
int main()
{
    printf("Hello World\n");
    return 0;
}
```

To the right of the code editor is a toolbar with a play icon (highlighted with a red box). Below the code editor is the Output panel, which displays the terminal output:

```
[Running] cd "d:\VSCode Workspace\" && gcc main.c -o main &&
Hello World

[Done] exited with code=0 in 0.254 seconds
```

The status bar at the bottom shows file information: Ln 7, Col 2, Spaces: 4, UTF-8, CRLF, C, Win32.

Structure of the C program

Basic structure of the C language

When you learn a language, you will find that almost all languages have a syntax that prints out the phrase "Hello World," which gives you a glimpse into learning the basic structure of the language. We will explore the basic structure of the C language using the code we wrote earlier.

- Basic components

- #include

- # is a preprocessor, i.e., a phrase to be preprocessed before compiling
 - Literally a command that needs to be processed before compiling
 - an action to add what exists to the right of this phrase before compiling

- stdio.h

- A file with the .h extension is a header file in C.
 - A collection of information about functions provided by the standard C library
 - Short for standard input and output header file
 - Contain library functions that deal with input and output

```
#include <stdio.h>

int main()
{
    printf("Hello World\n");
    return 0;
}
```

Structure of the C program

Basic structure of the C language

When you learn a language, you will find that almost all languages have a syntax that prints out the phrase "Hello World," which gives you a glimpse into learning the basic structure of the language. We will explore the basic structure of the C language using the code we wrote earlier.

- Basic components

- return 0;
 - Return a value when the function finishes
 - As technology evolves, we can implicitly return 0 without writing return 0.
 - The C language standard recommends writing return 0;.
- Semicolon (;)
 - The most important characters in the C language
 - At the end of all code, you must type it (it acts as a period).
 - Missing the semicolon results in an error.
 - Missing it is one of the most common sources of errors.

```
#include <stdio.h>
int main()
{
    printf("Hello World\n");
    return 0;
}
```

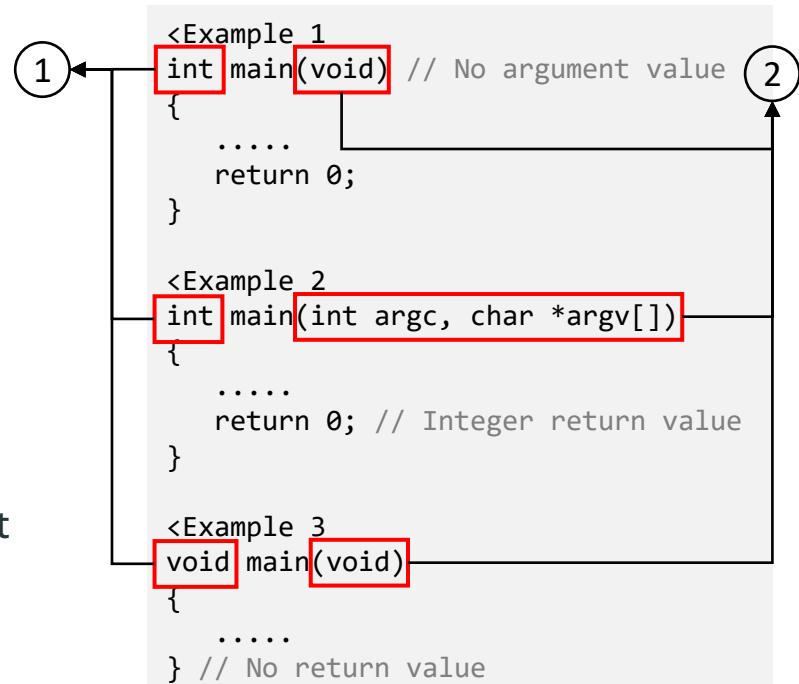
Structure of the C program

Basic structure of the C language

When you learn a language, you will find that almost all languages have a syntax that prints out the phrase "Hello World," which gives you a glimpse into learning the basic structure of the language. We will explore the basic structure of the C language using the code we wrote earlier.

- Basic components

- The function, main
 - The first function executed when a program written in C is executed
 - Only one exists within a program
 - ① Return value type
 - Returned as an integer (int type)
 - ② Argument value type (void means nothing)
 - Can take one of the following values as an argument
 - Argument count (argc : argument count)
 - Argument value (argv : argument value)
 - Environment variable value (envp : environment value)



Structure of the C program

Basic structure of the C language

When you learn a language, you will find that almost all languages have a syntax that prints out the phrase "Hello World," which gives you a glimpse into learning the basic structure of the language. We will explore the basic structure of the C language using the code we wrote earlier.

- Basic components

- Comment
 - Areas that have content but are not readable by the computer
 - Use them to comment any piece of code
 - Types of comment
 - One line comment : //
 - Multiline comment : /* */

```
/*
These comments can be multi-line.
1
2
3
4
*/
#include <stdio.h> // Headers

int main(void)
{
    printf("Hello World"); // call the function
    return 0;
}
```

C - variables and data types

Variables and basic data types

A variable is a place to store data in a program, and you must specify the data properties, such as the type and size of the data that can be stored in the variable.

- Variables

- A place to store values like numbers and letters.
- Organized by data type (①) and variable name (②)
- How to declare variables
 - How to just declare a variable

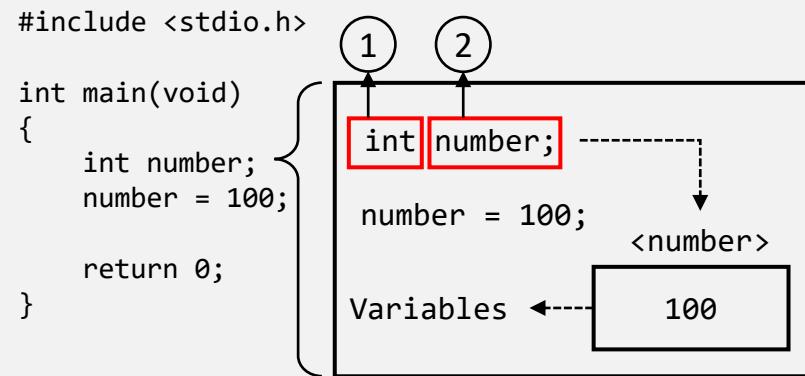
```
Type variable name, [variable name];
```

```
int number;  
number = 100;
```

- How to initialize a variable simultaneously with its declaration

```
Type variable name = initial value, [variable name  
= initial value];
```

```
int number = 100;  
int number1 = 200, number2 = 300;
```



C - variables and data types

Variables and basic data types

A constant, like a variable, represents memory where data can be stored. The difference between constants and variables is that the data stored in constants cannot be changed while the program is running.

- Constants

- Data that cannot be changed while the program is running
- Data types are also available for constants.

- Types of constants

- Literal constants
 - Constants with undefined names
- Symbolic constants
 - Constants with defined names
 - Use keywords such as const, define
 - Initialization is required when using constants

```
#include <stdio.h>

int main(void)
{
    //Symbolic constant names are implicitly uppercased
    //The rule is to show it.
    const int NUM = 100; // symbolic constant

    printf("%d\t",27); // literal constant
    // Error due to a constant change command to NUM
    // NUM = 200;
    printf("%d\n",NUM);

    return 0;
}

<Result>
27
100
```

C - variables and data types

Variables and basic data types

Programs use identifiers to distinguish variables, functions, etc., and identifiers cannot be words in keywords, which are special features of the C language.

- **Keywords**

- Words with special meanings that are pre-described to the compiler
- Also known as reserved words

C language keyword list

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

C - variables and data types

Variables and basic data types

Programs use identifiers to distinguish variables, functions, etc., and identifiers cannot be words in keywords, which are special features of the C language.

- Identifiers

- Unique names for identifying variables, constants, functions, etc.
- Identifier rules
 - Only alphabets (A-Z, a-z), numbers (0-9), and an underscore ("_") are allowed.
 - The first character cannot start with a number (0-9).
 - Case sensitive
 - Treat different cases as different identifiers
 - No whitespaces, no blank characters, and no special characters are allowed.
 - Keyword cannot be used as an identifier.
 - If the above is not followed, the compiler will not recognize the identifier.

C - variables and data types

Variables and basic data types

Programs use identifiers to distinguish variables, functions, etc., and identifiers cannot be words in keywords, which are special features of the C language.

- Identifiers

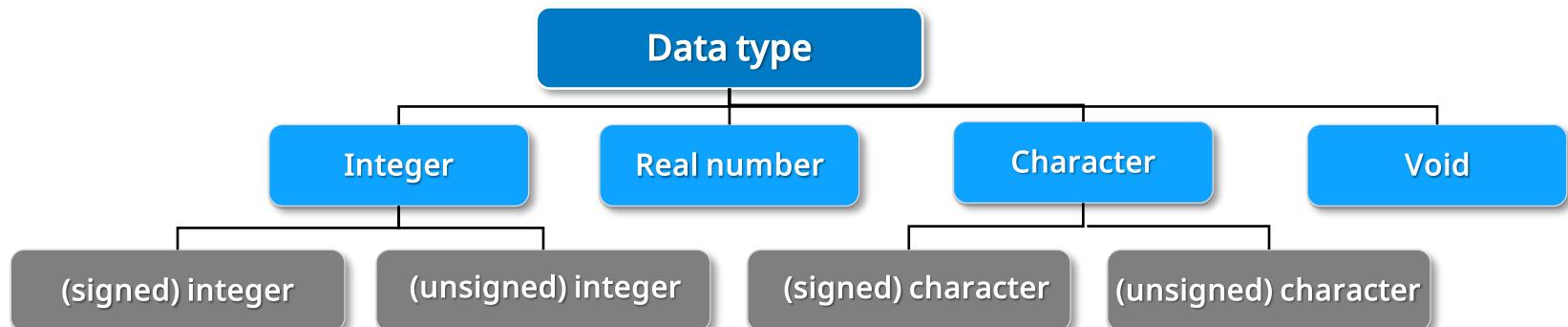
- Select all of the following that are available as variable names.
 - char ;
 - char a;
 - int number, Number;
 - float num1;
 - double num_@;
 - int num_1;
 - int long;

C - variables and data types

Variables and basic data types

A data type is a type of variable. Before you put a value into a variable, you must decide what kind of value it can hold, such as an integer, a real number, a character, or a string.

- Data type
 - Basic types are broadly divided into integer, real, and character types
 - Sign
 - Signed (variable with a sign) is the default format, while unsigned (variable without a sign) keywords are available.
 - Available only for integer and character data types
 - E.g., short expression range: -32,768 to 32,767 | unsigned short: 0 to 65,535



C - variables and data types

Variables and basic data types

A data type is a type of variable. Before you put a value into a variable, you must decide what kind of value it can hold, such as an integer, a real number, a character, or a string.

- Size by data type
 - The size of a data type depends on the operating system and the processing power of the computer.

Division	Data type	Size	Minimum	Maximum
Integer	short	2 bytes	-32,768	32,767
	unsigned short	2 bytes	0	65,535
	int	4 bytes	-2,147,483,648	2,147,483,647
	unsigned int	4 bytes	0	4,294,967,295
	long	4 bytes	-2,147,483,648	2,147,483,647
	unsigned long	4 bytes	0	4,294,967,295
Character type	char	1 byte	-128	127
	unsigned char	1 byte	0	255
Real numbers	float	4 bytes	-3.4×10^{-38} (7 significant digits)	$+3.4 \times 10^{38}$ (7 significant digits)
	double	8 bytes	-1.7×10^{-308} (15 significant digits)	$+1.7 \times 10^{308}$ (15 significant digits)
	long double	12 bytes	-1.7×10^{-308} (15 significant digits)	$+1.7 \times 10^{308}$ (15 significant digits)

C - variables and data types

Variables and basic data types

A data type is a type of variable. Before you put a value into a variable, you must decide what kind of value it can hold, such as an integer, a real number, a character, or a string.

- Size by data type
 - Different sizes for different data types on different operating systems
 - Different sizes for different data types based on data processing units (bits)

Operating system	char	short	int	long	pointer
32-bit Unix	1 byte	2 bytes	4 bytes	4 bytes	4 bytes
64-bit Unix	1 byte	2 bytes	4 bytes	8 bytes	8 bytes
64-bit Windows	1 byte	2 bytes	4 bytes	4 bytes	8 bytes

C - variables and data types

Variables and basic data types

- Variables and data types lab exercise
 - Source file - "data_type1.c"

```
#include <stdio.h>

int main(void)
{
    int a = 2147483648;
    short b = 65536;
    long c = 2147483648;

    printf("int : %d\n", a);
    printf("short : %hd\n", b);
    printf("long : %ld\n", c);
    return 0;
}
```

C - variables and data types

Variables and basic data types

- Data type size lab exercise
 - Source file - "data_type2.c"

```
#include <stdio.h>

int main(void)
{
    printf("short size : %d\n", sizeof(short));
    printf("int size : %d\n", sizeof(int));
    printf("long size : %d\n", sizeof(long));
    printf("char size : %d\n", sizeof(char));
    printf("float size : %d\n", sizeof(float));
    printf("double size : %d\n", sizeof(double));
    printf("long double size : %d\n", sizeof(long double));

    return 0;
}
```

C - operator

Operators

An operator is a symbol that tells the computer to execute a mathematical or logical command. C supports a wider range of operators compared to other programming languages.

- Arithmetic operators
 - Operators that perform numeric calculations

Division	Operator	Description	Syntax	Example
Polynomial operators	+	Addition of two operands	x + y	10 + 2 // 12
	-	Subtraction of two operands	x - y	10 - 2 // 8
	*	Multiplication of two operands	x * y	10 * 2 // 20
	/	The value of a quotient by dividing two operands	x/y	10 / 2 // 5
	%	The remainder of the division of two operands	x % y	10% 2 // 0
Unary operators	++	Increment operator - increase operand by 1	x++, ++x	++10 // 11
	--	Decrement operator - decrease operand by 1	x--, --x	--10 // 9
	+	Show a plus (positive) sign	+x	+10 // 10
	-	Show a minus (negative) sign	-x	-10 // -10

C - operator

Operators

An operator is a symbol that tells the computer to execute a mathematical or logical command. C supports a wider range of operators compared to other programming languages.

- Arithmetic operators lab exercise
 - Source file - "Arithmetic.c"

```
#include <stdio.h>

int main(void)
{
    int value1 = 20, value2 = 3;
    printf("<polynomial operator>\n");
    printf("+ : %d\n", value1 + value2);
    printf("- : %d\n", value1 - value2);
    printf("* : %d\n", value1 * value2);
    printf("/ : %d\n", value1 / value2);
    printf("% : %d\n", value1 % value2);

    return 0;
}
```

```
<polynomial operator>
+ : 23
- : 17
* : 60
/ : 6
% : 2
```

C - operator

Operators

An operator is a symbol that tells the computer to execute a mathematical or logical command. C supports a wider range of operators compared to other programming languages.

- Relational operators
 - Operators that compare operands to make either a true or false determination
 - True refers to any number other than zero, false refers to zero.

Operator	Description	Syntax	Example when true
<code>==</code>	Compare two operands if they are equal.	<code>x == y</code>	<code>20 == 20</code>
<code>!=</code>	Compare two operands to see if they are different.	<code>x != y</code>	<code>20 != 10</code>
<code>></code>	Compare if the left operand is greater than the right operand.	<code>x > y</code>	<code>20 > 10</code>
<code><</code>	Compare if the left operand is smaller than the right operand.	<code>x < y</code>	<code>10 < 20</code>
<code>>=</code>	Compare if the left operand is greater than or equal to the right operand.	<code>x >= y</code>	<code>20 >= 10</code>
<code><=</code>	Compare if the left operand is less than or equal to the right operand.	<code>x <= y</code>	<code>10 <= 20</code>

C - operator

Operators

An operator is a symbol that tells the computer to execute a mathematical or logical command. C supports a wider range of operators compared to other programming languages.

- Relational operators lab exercise
 - Source file - "Relational.c"

```
#include <stdio.h>

int main(void)
{
    int value1 = 20;
    int value2 = 20;

    printf("%d\n", value1 == value2);
    printf("%d\n", value1 != value2);
    printf("%d\n", value1 > value2);
    printf("%d\n", value1 <= value2);
}
```

```
<Result>
1
0
0
1
```

C - operator

Operators

An operator is a symbol that tells the computer to execute a mathematical or logical command. C supports a wider range of operators compared to other programming languages.

- Logical operators
 - Operators that perform logical operations (AND, OR, NOT)
 - True refers to any number other than zero, false refers to zero.

Operator	Description	Syntax	Example when true
&&	Perform a logical operation (AND) on two operands	$x \ \&\& \ y$	$x = 1$ and $y = 1$
	Perform a logical addition (OR) on two operands	$x \ \ y$	$x = 1$ or $y = 1$
!	Perform a logical negation (NOT) on two operands	$!x$	$x = 0$

C - operator

Operators

An operator is a symbol that tells the computer to execute a mathematical or logical command. C supports a wider range of operators compared to other programming languages.

- Logical operators lab exercise
 - Source file - "Logical.c"

```
#include <stdio.h>

int main(void)
{
    printf("%d\n", 1 && 0);
    printf("%d\n", 1 || 0);
    printf("%d\n", !1);
    printf("%d\n", !0);

    return 0;
}
```

<Result>
0
1
0
1

C - operator

Operators

An operator is a symbol that tells the computer to execute a mathematical or logical command. C supports a wider range of operators compared to other programming languages.

- Bitwise operators
 - Operators that perform bit-level operations on their operands

Operator	Description	Syntax	Example when true
&	Bit-level AND operations	$x \& y$	$5 \& 2 // 0$
	Bit-level OR operations	$x y$	$5 2 // 7$
^	Bit-level XOR operations	$x ^ y$	$5 ^ 2 // 7$
~	Bit-level NOT operations	$\sim x$	$\sim 5 // -6$
<<	Bit-level left shift operations	$x << y$	$5 << 2 // 20$

C - operator

Operators

An operator is a symbol that tells the computer to execute a mathematical or logical command. C supports a wider range of operators compared to other programming languages.

- Bitwise operators lab exercise
 - Source file - "Bitwise.c"

```
#include <stdio.h>

int main(void)
{
    printf("& : %d\n", 5 & 2);
    printf("| : %d\n", 5 | 2);
    printf("^ : %d\n", 5 ^ 2);
    printf("~ : %d\n", ~5);
    printf("<< : %d\n", 5 << 2);
    printf(">> : %d\n", 5 >> 2);

    return 0;
}
```

```
<Result>
& : 0
| : 7
^:7
~ : -6
<< : 20
>> : 1
```

C - operator

Operators

An operator is a symbol that tells the computer to execute a mathematical or logical command. C supports a wider range of operators compared to other programming languages.

- Assignment operators
 - Operators that assign the values of their operands or the result of an operation

Operator	Description	Syntax
=	Assign the value of the right operand to the left operand	$x = y$
+=	Add operands and assign the result to the left operand	$x += y, x = x + y$
--	Subtract operands and assign the result to the left operand	$x -= y, x = x - y$
*=	Multiply operands and assign the result to the left operand	$x *= y, x = x * y$
/=	Divide operands and assign the quotient of the result to the left operand	$x /= y, x = x / y$
%=	Divide operands and assign the remainder of the result to the left operand	$x %= y, x = x \% y$

C - operator

Operators

An operator is a symbol that tells the computer to execute a mathematical or logical command. C supports a wider range of operators compared to other programming languages.

- Assignment operators
 - Operators that assign the values of their operands or the result of an operation

Operator	Description	Syntax
<code><=</code>	Left-shift operands and assign the result to the left operand	<code>x <= y, x = x << y</code>
<code>>=</code>	Right-shift operands and assign the result to the left operand	<code>x >= y, x = x >> y</code>
<code>&=</code>	AND operands and assign the result to the left operand	<code>x &= y, x = x & y</code>
<code> =</code>	OR operands and assign the result to the left operand	<code>x = y, x = x y</code>
<code>^=</code>	XOR operands and assign the result to the left operand	<code>x ^= y, x = x ^ y</code>
<code><=</code>	Left-shift operands and assign the result to the left operand	<code>x <= y, x = x << y</code>

C - operator

Operators

An operator is a symbol that tells the computer to execute a mathematical or logical command. C supports a wider range of operators compared to other programming languages.

- Assignment operators lab exercise
 - Source file - "Assignment.c"

```
#include <stdio.h>

int main(void)
{
    int value1 = 1;

    printf("%d\n", value1 += 2);
    printf("%d\n", value1 -= 2);
    printf("%d\n", value1 *= 6);
    printf("%d\n", value1 /= 2);
    printf("%d\n", value1 %= 2);

    return 0;
}
```

```
<Result>
3
1
6
3
1
```

C - operator

Operators

An operator is a symbol that tells the computer to execute a mathematical or logical command. C supports a wider range of operators compared to other programming languages.

- Other operators

Name	Operator	Description	Syntax
Comma operator	,	Operator that lists operands of the same nature	int i, j;
Sizeof operator	sizeof	Operator that assigns the size of a variable or data type	sizeof(i), sizeof(char)
Cast operator	(type)	Operator that forces a change in the data type of an operand	b = (float) a
Conditional operator	? :	Operator that examines a conditional expression and outputs a result based on whether the condition is true or false.	c = (a>b) ? (a-b) : (b-a)
Address operator	&	Operator that returns the address of a variable	&number
Reference operator	*	A pointer to a variable	*address

C - operator

Operators

- C operator precedence
 - Operator precedence exists

Ranking	Operator type	Relevance
1	(suffix)++ (suffix)-- () [] . ->	Left to right
2	++(prefix) --(prefix) + - ! ~ (type) *(pointer) &(address) sizeof	Right to left
3	* / %	Left to right
4	+ -	Left to right
5	<< >>	Left to right
6	< <= > >=	Left to right
7	== !=	Left to right
8	&(AND)	Left to right
9	^	Left to right
10		Right to left
11	&&	Left to right
12		Left to right
13	? :	Right to left
14	= += -= *= /= %= <<= >>= &= ^= =	Right to left
15	,	Left to right

C - operator

Operators

- Practice questions
 - Compute the following formula.
 - $1 + 3 * 5 + 10 / 2$
 - $20 >> 1 + 1 ^ 3$
 - What is the result value of the following formula and the value of x after obtaining the result?
 - If $x = 1$, then $x++ + ++x + x++ = ?$

C - operator

Operators

- Answers to the practice questions
 - Compute the following formula.
 - $1 + 3 * 5 + 10 / 2$
 - $1 + (3 * 5) + (10 / 2) = 21$
 - $20 >> 1 + 1 ^ 3$
 - $20 >> (1 + 1) ^ 3 = 6$
 - What is the result value of the following formula and the value of x after obtaining the result?
 - If $x = 1$, then $x++ + ++x + x++ = ?$
 - $x++(1) + ++x(4) + x++(2) = 7$
 - The value of x : 4

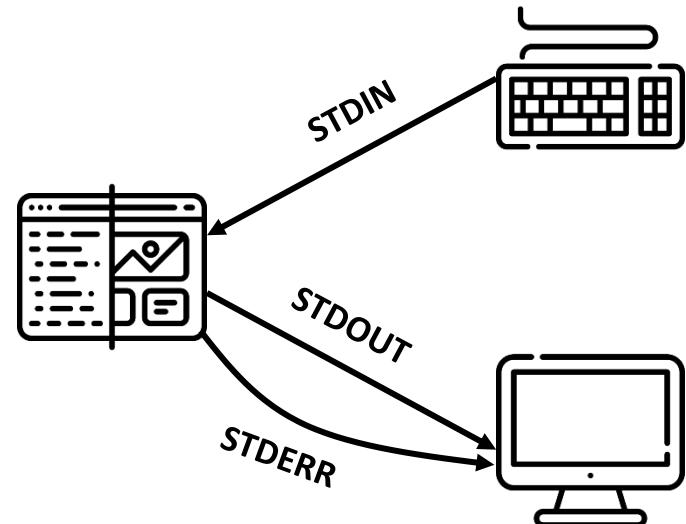
C - standard data input/output

Data input and output

Input refers to the entry of data into a program, while output refers to the display of data on the screen, in a file, or in other ways.

- Standard stream

- A predefined input/output path between a computer program and the computer environment
- Standard output (stdout) : the output stream that goes from a program to a monitor, etc.
 - Standard output functions : printf(), puts(), putchar(), etc.
- Standard input (stdin) : the input stream coming into a the program from a keyboard
 - Standard input functions : scanf(), gets(), getchar(), etc.
- Standard error (stderr) : stream of errors encountered by the program



C - standard data input/output

Data input and output

Input refers to the entry of data into a program, while output refers to the display of data on the screen, in a file, or in other ways.

- Escape sequence
 - Special characters used to change the state of a computer and its peripherals
 - These special characters are expressed by prefixing them with " \ " or " \\ "
 - Key escape characters

Escape sequence	Meaning	Description
\\	Backslash output	Literalize the backslash
\b	Backspace	Move the cursor back one space
\f	Form feed	Move the cursor to the beginning of the next page
\n	New line	Move the cursor to the beginning of the next line
\r	Carriage return	Move the cursor to the beginning of the current line
\t	Horizontal tab	Move the cursor to the next location on the Horizontal tab
\v	Vertical tabs	Move the cursor to the next location on the vertical tab

C - standard data input/output

Data input and output

Input refers to the entry of data into a program, while output refers to the display of data on the screen, in a file, or in other ways.

- Escape sequence lab exercise
 - Source file - "Escape_Sequence1.c" - Build after writing the code below.

```
#include <stdio.h>

int main(void)
{
    printf("[tab] : <\t \\\t>\n");
    printf("[backspace] : < No\b\bYes>\n");
    printf("[special character] : <input the \"or \\ you must input \\.>\n");
    printf("[carriage return]\n<it is a boy>\r<he\n");
    printf("[vertical tab] : <hi\vhello>\n");
    printf("[form feed] : <hi\fhello>\n");

    return 0;
}
```

C - standard data input/output

Data input and output

Input refers to the entry of data into a program, while output refers to the display of data on the screen, in a file, or in other ways.

- Escape sequence lab exercise results
 - Source file - "Escape_Sequence1.c" - Build after writing the code below
 - Console line breaks appear when keyboard input is not possible, such as vertical tabs, form feeds, etc.
 - This is caused by the console not recognizing the escape sequence → Not a compiler recognition issue.

```
[tab] : <           \t>
[backspace] : < Yes>
[special character] : <input the "or \ you must input \.>
[carrage return]
<he is a boy>
[vertical tab] : <hi
                  hello>
[form feed] : <hi
hello>
```

C - standard data input/output

Data input and output

Input refers to the entry of data into a program, while output refers to the display of data on the screen, in a file, or in other ways.

- Format specifier
 - Specify the data type during the data input and output process
 - Without proper identification of the format specifier, the program may output unintended data.

Specifier	Description	Example
%c	One character (char)	a
%s	String	Hello World
%d	Signed decimal integer (short, int) - %ld(long), %lld(long long)	123 or -123
%f	Fixed-point real number (up to 6 decimals) (float)	0.123456
%lf	Fixed-point real number (double) - %Lf(long double)	0.123456
%o	Unsigned octal integer	123 -> 173
%u	Unsigned decimal integer	123
%x or %X	Unsigned hexadecimal integer (use lowercase or uppercase letters)	123 -> 7b or 7B
%e or %E	Floating-point real numbers (e or E -notation)	0.00001234 -> 1.234e-05
%%	Print a percentage (%%) symbol	%

C - standard data input/output

Data input and output

Input refers to the entry of data into a program, while output refers to the display of data on the screen, in a file, or in other ways.

- Format specifier lab exercise and the result
 - Source file - "FormatSpecifier1.c" - Build after writing the code below.

```
#include <stdio.h>

int main(void)
{
    printf("%d + %d = %d\n",10,20,10 + 20); // Format the value with a format specifier
    printf("%f\n",10.0 + 20.0);
    printf("%d %c\n",97,97);

    return 0;
}

<Result>
10 + 20 = 30
30.000000
97 a
```

C - standard data input/output

Data input and output

Input refers to the entry of data into a program, while output refers to the display of data on the screen, in a file, or in other ways.

- Format specifier lab exercise
 - Source file - "FormatSpecifier2.c" - Build after writing the code below

```
#include <stdio.h>

int main(void)
{
    printf("Numeric output: %d %d \n", 'a', 65);
    printf("Character output: %c %c \n", 'a', 65);
    printf("nDecimal output: %d %x %o %#x %#o \n", 100, 100, 100, 100, 100, 100);
    printf("Real number : %4.2f %2.2e\n", 3.1416, 3.1416);
    printf("Width of integer output: \n %+5d\n %+5d\n %+5d\n %+5d\n", 1, 10, 100, 1000); //5 digits fixed, blank space
    printf("Degree of real number output: \n %.4f\n %.5f\n", 3.1415, 3.1415); //Print to 4 or 5 decimal places
    printf("<%s>\t <.4s>\t <%4.s>\n", "A string", "A string", "A string", "A string");

    return 0;
}
```

C - standard data input/output

Data input and output

Input refers to the entry of data into a program, while output refers to the display of data on the screen, in a file, or in other ways.

- Format specifier lab exercise result
 - Source file - "FormatSpecifier2.c" - Build after writing the code below.

```
<Result>

Number output: 97 65
Character output: a A
nDecimal output: 100 64 144 0x64 0144
Real number: 3.14 3.14e+00
Width of integer output:
    +1
    +10
    +100
    +1000
Degree of real number output:
    3.1415
    3.14150
<A string>      <A st>  <      >
```

C - standard data input/output

Data input and output

- printf function
 - Standard output function that prints output data in the specified format
 - Function format: printf("[sentence or format specifier to print]",[variable or "constant"],[variable or "constant"],...);
 - Format specifier usage: %[flag][width][.precision][length]format specifier

```
printf("%010.2f\n", 1.2f);
Result: 0000001.20
```

Flag	Description
-	Left justification
+	Print a plus (+) sign for a positive number and a minus (-) sign for a negative number.
Space	If positive, print no sign and leave space, if negative, print a minus (-) sign.
#	Prefix numbers with 0, 0x, and 0X specifiers to match their numeric system.
0	Fill the rest of the space in the output width with zeros.
Width	Description
Number	Print as wide as the number specified, with real numbers up to and including . (dots) and e+ in the width.
Precision	Description
. number	Print numbers up to a specified number of decimal places

C - standard data input/output

Data input and output

- The printf function lab exercise

```
#include <stdio.h>

int main(void)
{
    printf(" The result using %d : |%d|\n", 123);
    printf(" The result using %%7d : |%7d|\n", 123);
    printf(" The result using %%+7d : |%+7d|\n", 123);
    printf(" The result using %%-7d : |%-7d|\n\n", 123);

    printf(" The result using %%f : |%f|\n", 1.23);
    printf(" The result using %%.1f : |%.1f|\n", 1.23);
    printf(" The result using %%7.2f : |%7.2f|\n", 1.23);

    return 0;
}

<Result>
The result using %d : |123|
The result using %%7d : |      123|
The result using %%+7d : |     +123|
The result using %%-7d : |123      |

The result using %%f : |1.230000|
The result using %%.1f : |1.2|
The result using %%7.2f : |    1.23|
```

C - standard data input/output

Data input and output

- `scanf` function
 - Standard input function that takes in data and stores it in a variable
 - Function format: `scanf("[format specifier] [format specifier]", &[variables], &[variables],);`
 - An `&` sign : address operator, means to store the received data at the address of the variable.

```
#define _CRT_SECURE_NO_WARNINGS // Prevent compilation errors caused by scanf security warnings
#include<stdio.h>

int main(void)
{
    int number;

    printf("Please enter a number : ");
    scanf("%d", &number);

    printf("The number you entered is %d.\n", number);

    return 0;
}

<Result>
Please enter a number : 100
The number you entered is 100.
```

C - standard data input/output

Data input and output

- The scanf function lab exercise

```
#define _CRT_SECURE_NO_WARNINGS // Prevent compilation errors caused by scanf security warnings
#include<stdio.h>

int main(void)
{
    float num01;
    double num02;

    printf("Please enter two real numbers : ");
    scanf("%f %f", &num01, &num02);
    printf("The two real numbers you entered are %f and %f.\n", num01, num02);
    printf("Of the two real numbers entered, a completely different value was stored as the second double type variable.\n\n");

    printf("Once again, please enter two real numbers : ");
    scanf("%f %lf", &num01, &num02);
    printf("The two real numbers you entered are %f and %f.\n", num01, num02);
    printf("This time, both real numbers were saved correctly.\n");
    return 0;
}

<Result>
Please enter two real numbers : 1.2 3.4
The two real numbers you entered are 1.200000 and 0.000000.
Of the two real numbers entered, a completely different value was stored as the second double type variable.

Once again, please enter two real numbers : 1.2 3.4
The two real numbers you entered are 1.200000 and 3.400000.
This time, both real numbers were saved correctly.
```

C - standard data input/output

Data input and output

- Practice questions
 - Use the results on the right to complete the code on the left.
 - Source file - "practice1.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int score_kr, score_math;

    printf("Please enter your grades (Korean, math): ");
    scanf("%d %d", _____);

    printf("[Korean]\tGrade_____ ", _____);
    printf("[Math]\tGrade_____ ", _____);
    return 0;
}
```

<Result>
Please enter your grades (language arts, math): 80 90
[Korean] Grade 80
[Math] Grade 90

- Write a code using the scanf function to take two numbers as input and print the following.
 - Source file - "practice2.c"

```
<Result>
Please enter two numbers >> 10.0 3.0
10 / 3 = 3.333
```

C - standard data input/output

Data input and output

- puts function

- Function that prints strings
- Function format: puts("[variable or string]");
- The puts function automatically appends a newline.
 - No need to append "\n"

- gets function

- Function that takes strings as input
- Function format : gets([variable]);
- Save until a newline in a string
- NULL character("\0") appended to end of string

```
// Source file - "puts_gets1.c"
#include <stdio.h>

int main(void)
{
    // variable that takes a string of 30 bytes as input
    char value[30];
    /* Because the puts function has newlines,
       it accepts input from the following lines. */
    puts("Please enter a number : ");
    gets(value);
    puts(value);

    return 0;
}

<Result>
Please enter a number :
20000
20000
```

C - standard data input/output

Data input and output

- putchar function
 - Function that prints characters
 - Function format : putchar("[variable or character]");

- getchar function
 - Function that takes character input
 - Function type : getchar();
 - Also used to initialize input values

```
// Source file - "putchar_getchar1.c"
#include <stdio.h>

int main(void)
{
    char str;

    printf("Please enter a character : ");
    str = getchar();

    putchar(str);
    putchar('A');

    return 0;
}

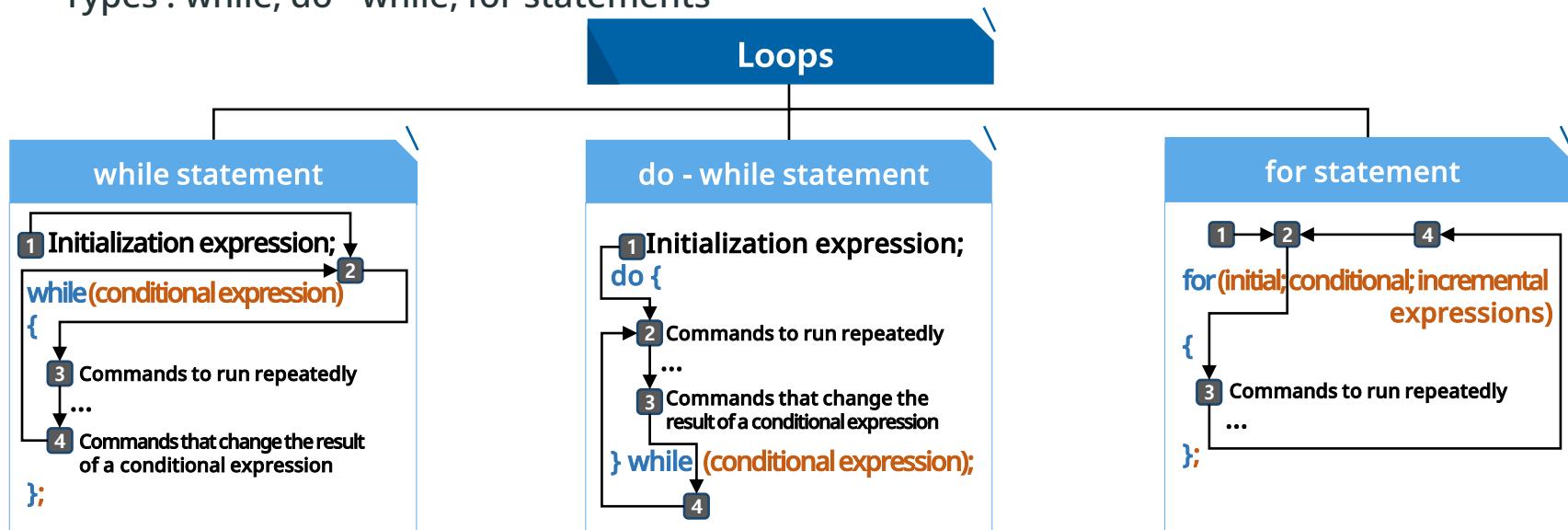
<Result>
Please enter a letter : a
```

C - control statements

Loops and conditional statements

A loop is a statement that tells a program to execute the same command a certain number of times. Because programs often contain repetitive code, it is one of the most commonly used control statements.

- Features and types of loops
 - Programming grammar to perform a specific task repeatedly
 - Use when working with constant rules and variations
 - Types : while, do - while, for statements



C - control statements

Loops and conditional statements

A loop is a statement that tells a program to execute the same command a certain number of times. Because programs often contain repetitive code, it is one of the most commonly used control statements.

- while statement

- while statement format : while([conditional statement])
- End the iteration if it consists only of conditional statements and the conditional statements are not true.
- The initial value exists outside the while statement
- Change starting condition exists inside the while statement
 - Create an infinite loop if it doesn't exist inside
 - If there is a break or return, even if the change starting condition doesn't exist inside, the statement can avoid getting stuck in an infinite loop.

```
#include <stdio.h>

int main(void)
{
    int value = 1; // Initial value
    while(value < 3) // Conditional statement
    {
        printf("%dth iteration\n",value);
        ++value; // Change starting condition
    }
    return 0;
}

<Result>
1st iteration
2nd iteration
```

C - control statements

Loops and conditional statements

A loop is a statement that tells a program to execute the same command a certain number of times. Because programs often contain repetitive code, it is one of the most commonly used control statements.

- while statement example
 - Implement a program that prints the sum of 1 through 9.
 - Source file - "while1.c"

```
#include <stdio.h>

int main(void)
{
    int value = 1;
    int result = 0;

    while(value < 10)
    {
        result += value;
        ++value;
    }
    printf("%d\n", result);
    return 0;
}
```

C - control statements

Loops and conditional statements

A loop is a statement that tells a program to execute the same command a certain number of times. Because programs often contain repetitive code, it is one of the most commonly used control statements.

- do - while statement

- Consist of do { }while ([conditional statement])
- End the loop if the conditional statement is not true after the first run
- Initial values exist outside of the do - while statement
- Change starting condition exists inside the do - while statement
 - Create an infinite loop if it doesn't exist inside
 - If there is 'break,' or 'return,' even if the change starting condition doesn't exist inside, the statement can avoid getting stuck in an infinite loop.

```
#include <stdio.h>

int main(void)
{
    int value = 0; // Initial value
    do
    {
        printf("%dth iteration\n", value);
        ++value; // Change starting condition
    } while(value < 1);

    return 0;
}

<Result>
0th iteration
1th iteration
```

C - control statements

Loops and conditional statements

A loop is a statement that tells a program to execute the same command a certain number of times. Because programs often contain repetitive code, it is one of the most commonly used control statements.

- do - while statement example
 - Implement a program that prints the sum of 1 through 9.
 - Source file - "do_while1.c"

```
#include <stdio.h>

int main(void)
{
    int value = 1;
    int result = 0;

    do
    {
        result += value;
        ++value;
    } while (value < 10);

    printf("%d\n", result);

    return 0;
}
```

C - control statements

Loops and conditional statements

A loop is a statement that tells a program to execute the same command a certain number of times. Because programs often contain repetitive code, it is one of the most commonly used control statements.

- for statement

- Consist of for([initial value]; [conditional statement]; [change starting condition])
- End the iteration if the conditional statement is not true
- There can be more than one change starting condition, but there can only be one conditional statement.
- Compare conditional statements again when the value changes based on a change starting condition

```
#include <stdio.h>

int main(void)
{
    int num;
    for(num=1; num<5; num++)
    {
        printf("%dth hello world output\n", num);
    }

    return 0;
}

<Result>
1th hello world output
2th hello world output
3th hello world output
4th hello world output
```

C - control statements

Loops and conditional statements

A loop is a statement that tells a program to execute the same command a certain number of times. Because programs often contain repetitive code, it is one of the most commonly used control statements.

- Example 1 of the for statement
 - Implement a program that prints 1 to 10.
 - Source file - "for1.c"

```
#include <stdio.h>

int main(void)
{
    int number;

    for (number = 1; number <= 10; number++)
    {
        printf("%d\t", number);
    }
    printf("\n");

    return 0;
}
```

C - control statements

Loops and conditional statements

A loop is a statement that tells a program to execute the same command a certain number of times. Because programs often contain repetitive code, it is one of the most commonly used control statements.

- Example 2 of the for statement
 - Implement a program that draws a star.
 - Source file - "for2.c"

```
#include <stdio.h>

int main(void)
{
    int i, j;

    for (i = 1; i <= 10; i++)
    {
        for (j = 1; j <= i; j++)
        {
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

C - control statements

Loops and conditional statements

- Practice questions on the while statement

- Implement a program to find the instantaneous number at which the numbers added together in order from 1 are greater than or equal to the number entered.
 - Source file - "while2.c"

```
<Result1>
Please enter a number : 20
Correct answer : 6
```

- Implement a program that takes an unlimited number of integers and terminates when a negative number is entered.
 - Source file - "while3.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int number = 0;
    while(______)
    {
        printf("Please enter a number : ");
        scanf("%d", &number);

        _____
    }
    return 0;
}
```

```
<Result>
Please enter a number : 20
The number you entered is 20.
Please enter a number : 10
The number you entered is 10.
Please enter a number : 5
The number you entered is 5.
Please enter a number : 4
The number you entered is 4.
Please enter a number : -1
```

C - control statements

Loops and conditional statements

- Answers to the practice questions on the while statement
 - Implement a program to find the instantaneous number at which the numbers added together in order from 1 are greater than or equal to the number entered.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int number, count=1, result=1;

    printf("Please enter a number : ");
    scanf("%d", &number);

    while (result < number)
    {
        ++count;
        result += count;
    }
    printf("Correct answer : %d\n", count);
    return 0;
}
```

C - control statements

Loops and conditional statements

- Answers to the practice questions on the while statement
 - Implement a program that takes an unlimited number of integers and terminates when a negative number is entered.

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int number = 0;
    while (number >= 0)
    {
        printf("Please enter a number : ");
        scanf("%d", &number);
        if (number < 0) break;
        printf("The number you entered is %d.\n", number);
    }
    return 0;
}
```

C - control statements

Loops and conditional statements

- Practice question on the do - while statement
 - Replace a while statement with a do-while statement but do it in a way that you can get the same action and result.
 - Source file - "do_while2.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int cnt = 0, sum = 0;
    int num, avg;

    while (1)
    {
        printf("Please enter an integer : ");
        scanf("%d", &num);
        if (num == 0) break;
        if (num % 2 == 1) continue;
        sum += num;
        cnt++;
    }
    avg = sum / cnt;
    printf("The number of even numbers entered is %d\n", cnt);
    printf("The sum of the entered even numbers is %d and the average is %d\n", sum, avg);
    return 0;
}
```

C - control statements

Loops and conditional statements

- Answer to the practice question on the do - while statement
 - Replace while statements with do-while statements to achieve the same behavior and results
 - Source file - "do_while2.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int cnt = -1, sum = 0;
    int num, avg;
    do
    {
        printf("Please enter an integer : ");
        scanf("%d", &num);
        if (num % 2 == 0)
        {
            sum += num;
            cnt++;
        }
    } while (num != 0);
    avg = sum / cnt;
    printf("The number of even numbers entered is %d\n", cnt);
    printf("The sum of the entered even numbers is %d and the average is %d\n", sum, avg);
    return 0;
}
```

C - control statements

Loops and conditional statements

- Practice question 1 on the for statement
 - Implement a program that prints the times table in the multiplication table that corresponds to the number entered.
 - Source file - "gugudan1.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int number;
    int i;

    printf("Please enter a number : ");
    scanf("%d", &number);

    for (____; ____; ____)
    {
        printf("%d * %d = %d\n", _____, _____, _____);
    }

    return 0;
}
```

C - control statements

Loops and conditional statements

- Practice question 2 on the for statement
 - Implement a program that draws a star and prints the following output (using three for statements).
 - Source file - "star1.c"

```
<Result>
*****
*****
*****
*****
*****
*****
*****
*****
*
```

C - control statements

Loops and conditional statements

- Answer to the practice question 1 on the for statement
 - Implement a program that prints the times table in the multiplication table that corresponds to the number entered.
 - Source file - "gugudan1.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int number;
    int i;

    printf("Please enter a number : ");
    scanf("%d", &number);

    for (i = 1;i < 10;i++)
    {
        printf("%d * %d = %d\n", number, i, number * i);
    }

    return 0;
}
```

C - control statements

Loops and conditional statements

- Answer to the practice question 2 on the for statement
 - Implement a program that draws a star and prints the following output (using three for statements).
 - Source file - "star1.c"

```
#include <stdio.h>

int main(void)
{
    int i, j, k;

    for (i = 1; i < 11; i++)
    {
        for (j = 1; j < i; j++)
        {
            printf(" ");
        }
        for (k = 1; k <= 11 - i; k++)
        {
            printf("*");
        }
        printf("\n");
    }

    return 0;
}
```

C - control statements

Loops and conditional statements

- Assignment

- A program that prints the multiplication table in a 3 X 3 table format (1 to 9 times).
 - Source file - "gugudan2.c"

```
1 * 1 = 1      2 * 1 = 2      3 * 1 = 3  
1 * 2 = 2      2 * 2 = 4      3 * 2 = 6  
1 * 3 = 3      2 * 3 = 6      3 * 3 = 9  
1 * 4 = 4      2 * 4 = 8      3 * 4 = 12  
1 * 5 = 5      2 * 5 = 10     3 * 5 = 15  
1 * 6 = 6      2 * 6 = 12     3 * 6 = 18  
1 * 7 = 7      2 * 7 = 14     3 * 7 = 21  
1 * 8 = 8      2 * 8 = 16     3 * 8 = 24  
1 * 9 = 9      2 * 9 = 18     3 * 9 = 27  
  
4 * 1 = 4      5 * 1 = 5      6 * 1 = 6  
4 * 2 = 8      5 * 2 = 10     6 * 2 = 12  
4 * 3 = 12     5 * 3 = 15     6 * 3 = 18  
4 * 4 = 16     5 * 4 = 20     6 * 4 = 24  
4 * 5 = 20     5 * 5 = 25     6 * 5 = 30  
4 * 6 = 24     5 * 6 = 30     6 * 6 = 36  
4 * 7 = 28     5 * 7 = 35     6 * 7 = 42  
4 * 8 = 32     5 * 8 = 40     6 * 8 = 48  
4 * 9 = 36     5 * 9 = 45     6 * 9 = 54  
  
7 * 1 = 7      8 * 1 = 8      9 * 1 = 9  
...  
...
```

C - control statements

Loops and conditional statements

A C program contains many commands that are executed in order from start to finish, so you must control the sequential flow of the program to achieve the desired result.

- Features and types of conditional statements
 - Statement that controls the execution of separate commands based on the results of a given conditional expression
 - Types: if, if-else, if-else if-else, and switch statements

```
if (conditional expression 1)
{
    The command to run if the result of conditional expression 1 is true;
}
else if (conditional expression 2)
{
    The command to run if the result of conditional expression 2 is true;
}
else
{
    The command to run if the result of conditional
    expression 1 is false and the result of conditional
    expression 2 is also false;
}
```

```
switch (condition value)
{
    case value1 :
        The command to run if the condition value is value1;
        break;
    case value2 :
        The command to run if the condition value is value2;
        break;
    ...
    default :
        The command to run if the condition value does not
        appear in any of the case clauses;
        break;
}
```

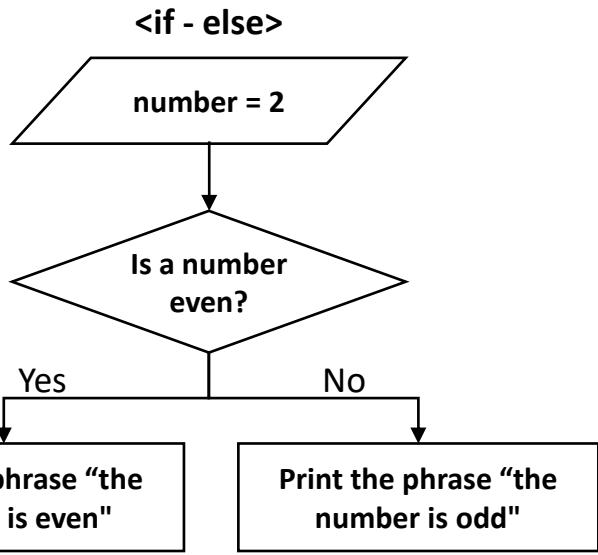
C - control statements

Loops and conditional statements

A C program contains many commands that are executed in order from start to finish, so you must control the sequential flow of the program to achieve the desired result.

- if - else statement

- Determine flow by evaluating whether a condition is true (or a non-zero value) or false (or zero)



```
#include <stdio.h>

int main(void)
{
    if(score >= 90) // Enclose conditional expression in ()
    {
        // Execute the code inside {} when the condition is met
        printf("Congratulations.");
        printf("It is A.\n");
    }
    else if (score >=80)
        // Can omit {} if it's a single line
        printf("It is B.\n");
    else
    {
        printf("It is C.\n");
    }

    // If the conditional expression is not met in the if statement,
    // move to the next else if statement or the else statement
    // else & else if statements can be added/subtracted as needed
    return 0;
}
```

C - control statements

Loops and conditional statements

A C program contains many commands that are executed in order from start to finish, so you must control the sequential flow of the program to achieve the desired result.

- if - else statement example

- Implement a program that takes an integer and prints a phrase if it is three or more digits.
 - Source file - "if-else.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int number;

    printf("Please enter a number : ");
    scanf("%d", &number);

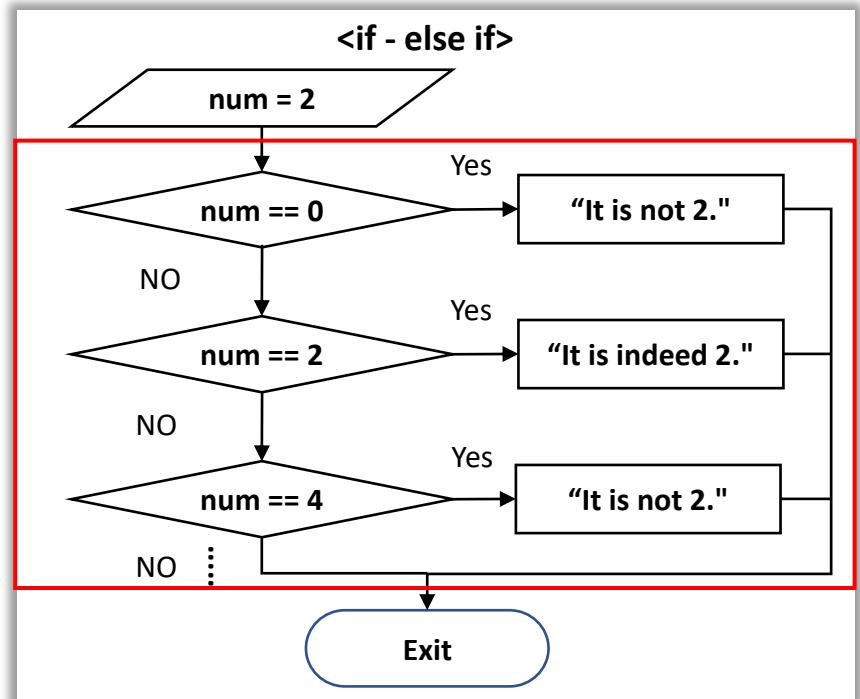
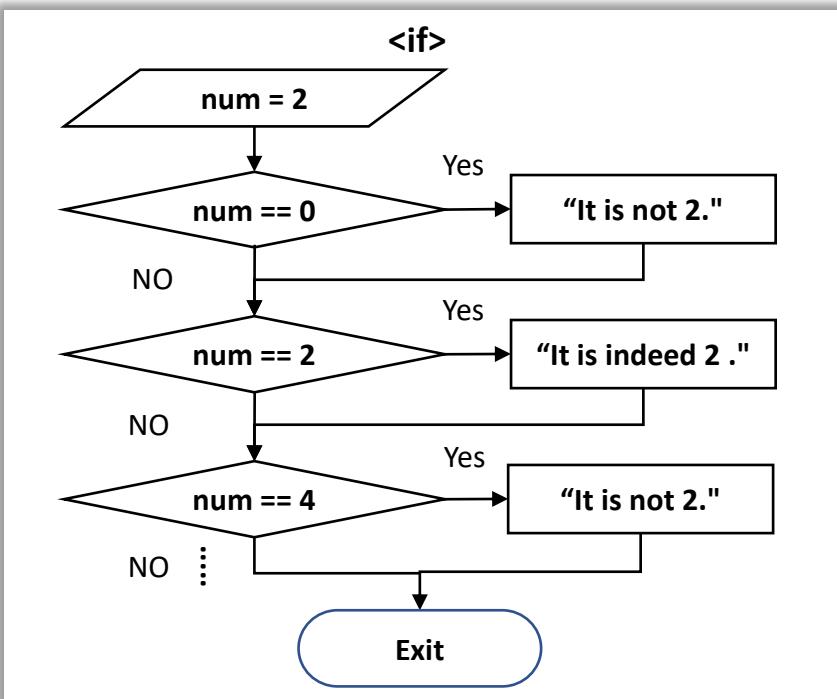
    if (number > 99)
    {
        printf("The number you entered is a three-digit number.\n");
    }
    return 0;
}
```

C - control statements

Loops and conditional statements

A C program contains many commands that are executed in order from start to finish, so you must control the sequential flow of the program to achieve the desired result.

- if - else if - else statement
 - Grammar for creating multiple conditions and executing commands that satisfy them



C - control statements

Loops and conditional statements

A C program contains many commands that are executed in order from start to finish, so you must control the sequential flow of the program to achieve the desired result.

- if - else if - else statement example

- Implement a program that takes integer inputs and distinguishes between zeros or odd/even.
 - Source file - "else-if.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int number;
    printf("Please enter a number : ");
    scanf("%d", &number);

    if (number == 0)
        printf("The number you entered is 0.\n");
    else if ((number % 2) == 0)
        printf("The number you entered is an even number.\n");
    else
        printf("The number you entered is an odd number.\n");
    return 0;
}
```

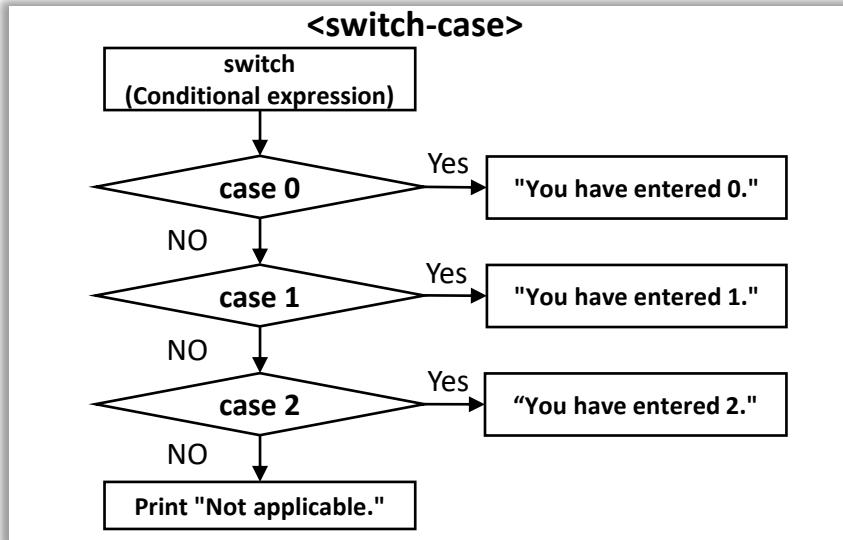
C - control statements

Loops and conditional statements

A C program contains many commands that are executed in order from start to finish, so you must control the sequential flow of the program to achieve the desired result.

- switch-case statement

- Used when there are multiple branches
- Run code in the case if the conditions are met
- Run default code if no conditions are met



```
#include <stdio.h>

int main(void)
{
    int num = 1;

    switch(num)
    {
        case 0:
            printf("You have entered 0.\n");
            break;
        case 1: // When num is 1
            printf("You have entered 1.\n");
            break;
        case 2: // When num is 2
            printf("You have entered 2.\n");
            break;
        default: // If no condition is satisfied
            printf("Not applicable.\n");
    }

    return 0;
}
```

C - control statements

Loops and conditional statements

- Example of the switch-case statement
 - Implementing a program
 - Source file - "switch-case2.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    char choice;
    printf("Do you want to run it? [y/n] ");
    scanf("%c", &choice);

    switch (choice)
    {
        case 'y':
            printf("This will be executed.\n");
            break;
        case 'n':
            printf("The execution was canceled.\n");
            break;
        default:
            printf("You have made a wrong selection.\n");
    }
    return 0;
}
```

C - control statements

Loops and conditional statements

- Practice question 1 on the if statement
 - Implement a program that subtracts a smaller number from a larger number.
 - Source file - "subtract.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int value1, value2;

    printf("Please enter two numbers : ");
    scanf("%d %d", &value1, &value2);

    if(_____)
        printf("It is %d - %d = %d.\n", value2, value1, value2-value1);
    else if(_____)
        printf("It is %d - %d = %d.\n", value1, value2, value1-value2);
    else
        printf("The two numbers are equal.\n");

    return 0;
}
```

C - control statements

Loops and conditional statements

- Practice question 2 on the if statement
 - Implement a program that tells you when you enter a year if it's a leap year.
 - Source file - "leap_year.c"

```
<Result>
Please enter a year : 2044
Year 2044 is a leap year.
```

C - control statements

Loops and conditional statements

- Practice question 3 on the switch-case statement
 - Implement a calculator program (arithmetic operations).
 - Source file - "calculator.c"

```
<Result1>
Please enter a formula to calculate : 4 * 5
4 * 5 = 20

<Result2>
Please enter a formula to calculate : 100 - 24
100 - 24 = 76
```

C - control statements

Loops and conditional statements

- Answer to the practice question 1 on the if statement
 - Implement a program to subtract a smaller number from a larger number.
 - Source file - "subtract.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int value1, value2;

    printf("Please enter two numbers : ");
    scanf("%d %d", &value1, &value2);

    if(value1 < value2)
        printf("It is %d - %d = %d.\n", value2, value1, value2-value1);
    else if(value1 > value2)
        printf("It is %d - %d = %d.\n", value1, value2, value1-value2);
    else
        printf("The two numbers are equal.\n");

    return 0;
}
```

C - control statements

Loops and conditional statements

- Answer to the practice question 2 on the if statement
 - Implement a program that tells you when you enter a year if it's a leap year.
 - Source file - "leap_year.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int year;

    printf("Please enter a year : ");
    scanf("%d", &year);

    if ((year % 4) == 0 && (year % 100) != 0 || (year % 400) == 0)
        printf("Year %d is a leap year.\n", year);
    else
        printf("Year %d is a normal year.\n", year);

    return 0;
}
```

C - control statements

Loops and conditional statements

- Answer to the practice question 3 (1/2) on the switch-case statement
 - Implement a calculator program (arithmetic operations).
 - Source file - "calculator.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    int value1;
    int value2;
    char operator;

    printf("Please enter a formula to calculate : ");
    scanf("%d %c %d", &value1, &operator, &value2);

    switch (operator)
    {
        case '+':
            printf("%d + %d = %d\n", value1, value2, value1 + value2);
            break;
        case '-':
            printf("%d - %d = %d\n", value1, value2, value1 - value2);
            break;
    }
}
```

C - control statements

Loops and conditional statements

- Answer to the practice question 3 (1/2) on the switch-case statement
 - Implement a calculator program (arithmetic operations).
 - Source file - "calculator.c"

```
case '*':  
    printf("%d * %d = %d\n", value1, value2, value1 * value2);  
    break;  
case '/':  
    printf("%d / %d = %d\n", value1, value2, value1 / value2);  
    break;  
default:  
    printf("Invalid formula.\n");  
}  
return 0;  
}
```

C - control statements

Loops and conditional statements

- Assignment
 - Implement a program that takes scissors (0), rock (1), paper (2) as input and plays the game scissors rock paper against the user.
 - Source file - "game.c"
 - Program implementation considerations
 - Computer randomly picks one of scissors (0), rock (1), paper (2) (using rand and srand functions) and plays against the user.
 - Rematch of something other than scissors (0), rocks (1), paper (2) is entered.

```
<Result>
Enter scissors (0), rock (1), paper (2): 3
Enter scissors (0), rock (1), paper (2): -1
Enter scissors (0), rock (1), paper (2): 2
You've lost.
```

C - control statements

Loops and conditional statements

Normally, when you enter a loop by checking a conditional expression, all statements in the loop are executed until the next conditional expression is checked. However, the continue and break statements allow you to control the flow of these common loops.

- Other control statements
 - continue statement
 - Skip the rest of that loop and jump to the next judgment of the conditional statement
 - Often used when you want to throw an exception for a specific condition within a loop.

```
#include <stdio.h>

int main(void)
{
    int i;
    int except_num = 3;

    for (i = 1; i <= 100; i++)
    {
        if (i % except_num == 0)
            continue;
        printf("%d ", i);
    }
    return 0;
}
```

C - control statements

Loops and conditional statements

Normally, when you enter a loop by checking a conditional expression, all commands in the loop are executed until the next conditional expression is checked. However, the continue and break statements allow you to control the flow of these common loops.

- Other control statements
 - break statement
 - Use it inside a loop to fully exit the iteration, and then run the command immediately after the iteration
 - Used to fully exit a loop, regardless of the conditional expression result inside the loop

```
#include <stdio.h>

int main(void)
{
    int start_num = 1;
    int end_num = 10;
    int sum = 0;
    while (1)
    {
        sum += start_num;
        if (start_num == end_num) break;
        start_num++;
    }
    return 0;
}
```

C - functions

Functions

A function in programming is a set of independently created pieces of code that carry out a specific task for a single, specific purpose. A C program consists of functions which are used to achieve the objectives of the program.

- Why do we use functions?

- Reusable code
- Reduce program complexity
- More efficient as programs become more modular

- Types of functions

- Library functions
 - Predefined functions in a programming language
- User-defined functions
 - User-defined arbitrary functions

```
#include <stdio.h>

int bigNum(int num01, int num02)
{
    if (num01 >= num02)
        return num01;
    else
        return num02;
}

int main(void)
{
    int result;
    result = bigNum(3, 5); // Calling of the function
    printf("The greater of the two numbers is %d.\n", result);

    result = bigNum(3, 1); // Calling of the function
    printf("The greater of the two numbers is %d.\n", result);

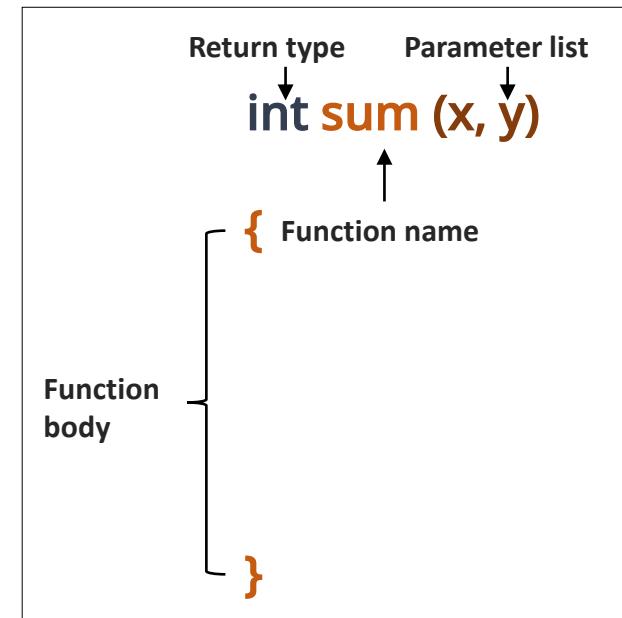
    result = bigNum(7, 5); // Calling of the function
    printf("The greater of the two numbers is %d.\n", result);
    return 0;
}
```

C - functions

Functions

A function in programming is a set of independently created pieces of code that carry out a specific task for a single, specific purpose. A C program consists of functions which are used to achieve the objectives of the program.

- The function definition
 - Return type
 - Specify the data type the function returns when it's finished.
 - It can return zero (void type) or one value
 - Function name
 - Specify a name to call the function
 - Parameter list (parameters)
 - Specify variables to store the values of arguments passed to the function when it is called
 - Multiple arguments can be passed in a function call
 - Function body
 - A compound statement performing the function's unique task



C - functions

Functions

A function in programming is a set of independently created pieces of code that carry out a specific task for a single, specific purpose. A C program consists of functions which are used to achieve the objectives of the program.

- The function definition
 - To define a function only, place it above the main function

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
// Define a custom function
int sum(int value1, int value2)
{
    return value1 + value2;
}

int main(void)
{
    int number1, number2, result;

    printf("Please enter two numbers : ");
    scanf("%d %d",&number1,&number2);

    result = sum(number1, number2);
    printf("Total: %d\n",result);
    return 0;
}
```

A function in programming is a set of independently created pieces of code that carry out a specific task for a single, specific purpose. A C program consists of functions which are used to achieve the objectives of the program.

- The prototype declaration for a function
 - Declare to the compiler before being used

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
int sum(int value1, int value2); // Declare a function

int main(void)
{
    int number1, number2, result;

    printf("Please enter two numbers : ");
    scanf("%d %d",&number1,&number2);

    result = sum(number1, number2);
    printf("Total: %d\n",result);
    return 0;
}
int sum(int value1, int value2)
{
    return value1 + value2;
}
```

The location of a variable's declaration in C determines its valid range, memory return time, initialization state, and storage location. Based on these characteristics, variables in C can be classified as follows.

- Effective range of a variable (variable scope)
 - Local variable : a variable declared within a "block"
 - Variables are valid only within the block in which they are declared, and are removed from memory when the block ends.
 - Global variable : a variable declared outside of a function
 - Accessible from anywhere in the program, and removed from memory when the program exits
 - Static variable : a variable declared with the static keyword.
 - Have the characteristics of both local and global variables
 - Like global variables, initialized only once and removed from memory upon exit
 - Accessible only within the function, like a local variable
 - Register variable : a variable declared with the register keyword when declaring a local variable.
 - Stored in the register memory of the CPU for quick access

C - functions

Functions

The location of a variable's declaration in C determines its valid range, memory return time, initialization state, and storage location. Based on these characteristics, variables in C can be classified as follows.

- Types of variables

Variable type	Keyword	Declaration location	Valid range
Local variable	auto	Inside a function/block	Inside a function/block
Global variable	extern	Outside of a function	Program-wide
Static variable	static	Inside a function/block	Inside a function/block
Register variable	register	Inside a function/block	Inside a function/block

Variable type	The time when memory is destroyed	Default value	Save to
Local variable	At the end of the function	Not initialized	Stack area
Global variable	At the end of the program	Reset to 0	Data area
Static variable	At the end of the program	Reset to 0	Data area
Register variable	At the end of the function	Not initialized	CPU register

C - functions

Functions

The location of a variable's declaration in C determines its valid range, memory return time, initialization state, and storage location. Based on these characteristics, variables in C can be classified as follows.

- Recursive call
 - Refer to a function that calls itself from within a function
 - Must contain a command that changes the condition to break the recursive call within a function

```
#include <stdio.h>

int sum(int n) {
    int i;
    int result = 0;
    for (i = 1; i <= n; i++)
    {
        result += i;
    }
    return result;
}

int main(void)
{
    printf("%d\n", sum(10));
    return 0;
}
```

```
#include <stdio.h>

int rSum(int n)
{
    if (n == 1)
    {
        return 1;
    }

    return n + rSum(n-1);
}

int main(void)
{
    printf("%d\n", rSum(10));
    return 0;
}
```

C - functions

Functions

- Example 1 (1/2) of the function declaration and definition
 - Source file - "function_declaration.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

void sum(int value1, char opt, int value2);
void sub(int value1, char opt, int value2);

int main(void)
{
    int number1, number2, result;
    char opt;

    printf("Please enter a formula : ");
    scanf("%d %c %d", &number1, &opt, &number2);

    switch (opt)
    {
        case '+':
            sum(number1, opt, number2);
            break;
```

- Example 1 (2/2) of the function declaration and definition
 - Source file - "function_declaration.c"

```
case '-':  
    sub(number1, opt, number2);  
    break;  
default:  
    printf("It is an invalid operation.\n");  
}  
}  
  
void sum(int value1, char opt, int value2)  
{ }  
    printf("%d %c %d = %d\n", value1, opt, value2, value1 + value2);  
}  
  
void sub(int value1, char opt, int value2)  
{ }  
    printf("%d %c %d = %d\n", value1, opt, value2, value1 - value2);  
}
```

C - functions

Functions

- Example 2 of the function declaration and definition
 - Source file - "recursive_function.c"

```
#include <stdio.h>
int factorial(int number);

int main()
{
    int number = 5;
    int result;
    result = factorial(number);
    printf("Result of 5! : %d \n", result);
    return 0;
}
int factorial(int n)
{
    if (n <= 1)
    {
        printf("%d\n", n);
        return 1;
    }
    else
    {
        printf("%d * ", n);
        return n * factorial(n - 1);
    }
}
```

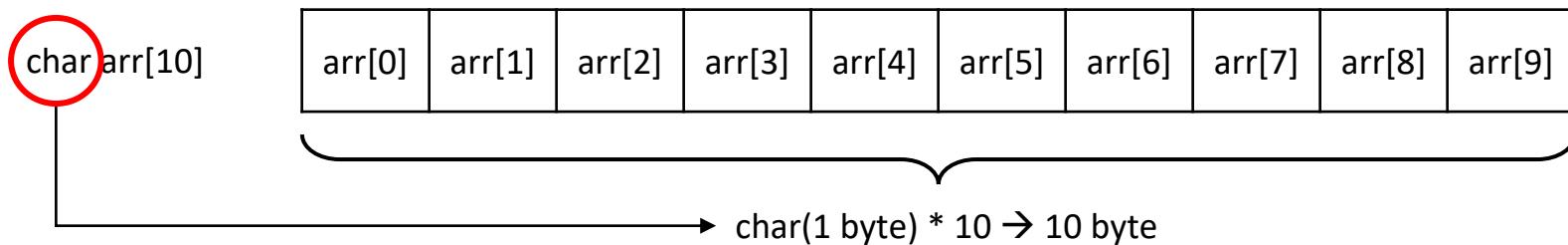
C - array and pointer

Arrays and pointers

An array is a structured data type that allows you to store data of the same type sequentially in memory. Data stored in an array is numbered (indexed) based on the starting point of the array, and data can be retrieved by number (index).

- Array

- Put multiple values in a single variable
 - Allow string input
- Store data sequentially in memory
- Allocate memory addresses equal to the size of the array when declaring an array
- Only the same data type can be declared.
- Array numbers (indices) start at 0.



C - array and pointer

Arrays and pointers

An array is a structured data type that allows you to store data of the same type sequentially in memory. Data stored in an array is numbered (indexed) based on the starting point of the array, and data can be retrieved by number (index).

- Array declaration and initialization
 - Store garbage value when declaring an array
 - An error occurs if more data is stored than the declared array size.

```
#include <stdio.h>

int main(void)
{
    int arr1[4]; // Uninitialized
    int arr2[4] = { 1,2,3,4 }; // Initialize to 1,2,3,4
    int arr3[4] = { 1 }; // Initialize to 1,0,0,0,0
    int arr4[] = { 1,2,3,4 }; // Initialize to 1,2,3,4

    printf("%d %d %d %d\n", arr1[0], arr1[1], arr1[2], arr1[3]);
    printf("%d %d %d %d\n", arr2[0], arr2[1], arr2[2], arr2[3]);
    printf("%d %d %d %d\n", arr3[0], arr3[1], arr3[2], arr3[3]);
    printf("%d %d %d %d\n", arr4[0], arr4[1], arr4[2], arr4[3]);

    return 0;
}
```

C - array and pointer

Arrays and pointers

An array is a structured data type that allows you to store data of the same type sequentially in memory. Data stored in an array is numbered (indexed) based on the starting point of the array, and data can be retrieved by number (index).

- Array example

- Source file - "array1.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    char buffer[12];
    int i;

    printf("Please enter a sentence : ");
    scanf("%[^\\n]s", buffer);

    printf("sentence : %s, \t size : %d\\n", buffer, sizeof(buffer));
    printf("%p %p %p\\n", &buffer[0], &buffer[1], &buffer[2]);
    for (i = 0; i < sizeof(buffer); i++)
    {
        printf("%c ", buffer[i]);
    }
    printf("\\n");
}
```

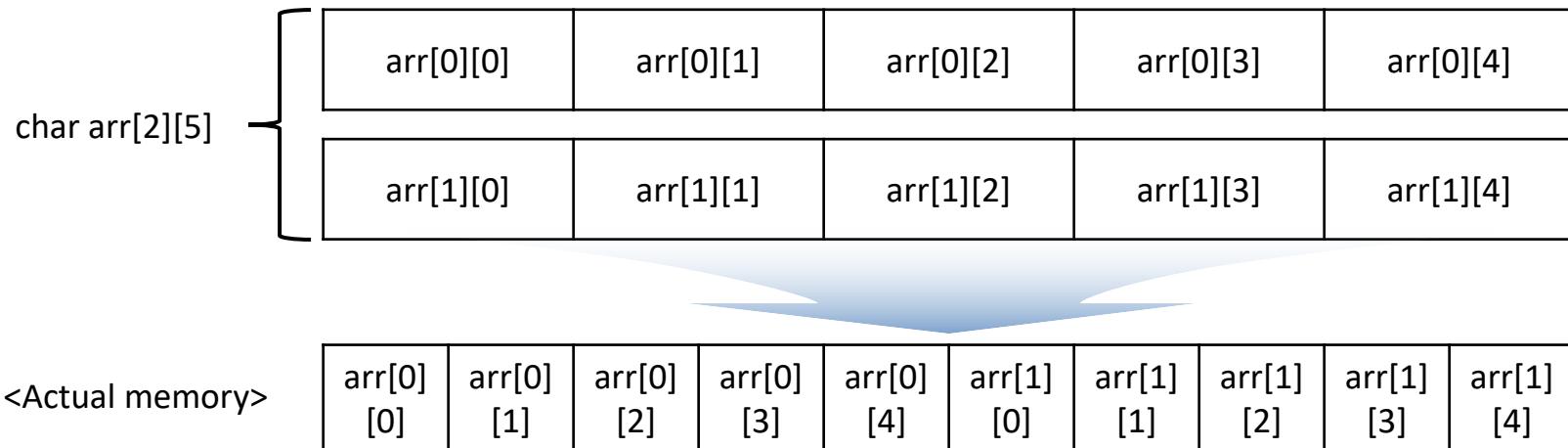
C - array and pointer

Arrays and pointers

An array is a structured data type that allows you to store data of the same type sequentially in memory. Data stored in an array is numbered (indexed) based on the starting point of the array, and data can be retrieved by number (index).

- N-dimensional array

- A one-dimensional array has N columns → char arr[N][M]
- For char arr[N][M], the actual size is $(N - 1) * (M - 1)$.
- Store linearly because memory doesn't have the concept of two dimensions
- Easy to write and program algorithms



C - array and pointer

Arrays and pointers

- Two-dimensional array example
 - Source file - "array2.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

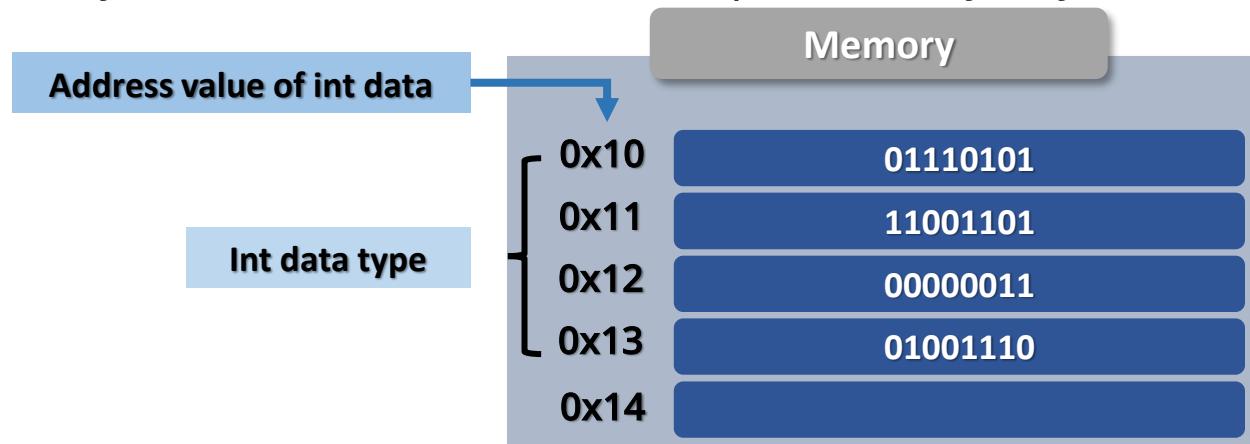
int main(void)
{
    int row, col;
    int number = 1;
    int arr[4][5];
    for (col = 0; col < 4; col++) {
        for (row = 0; row < 5; row++) {
            arr[col][row] = number;
            number++;
        }
    }
    for (col = 0; col < 4; col++) {
        for (row = 0; row < 5; row++) {
            printf("%2d\t", arr[col][row]);
        }
        printf("\n");
    }
    return 0;
}
```

C - array and pointer

Arrays and pointers

A pointer in C is a variable that stores an address in memory, also known as a pointer variable. Just as a char variable stores a character and an int variable stores an integer, a pointer stores an address value.

- The concept of a pointer
 - Understanding address values
 - The address value of the data refers to the starting address of the memory where the data is stored.
 - In C, expressed as a one-byte size in memory space
 - E.g., int data type is 4 bytes in size, and its address value points to only 1 byte of the starting address.



Source : TCP school homepage

C - array and pointer

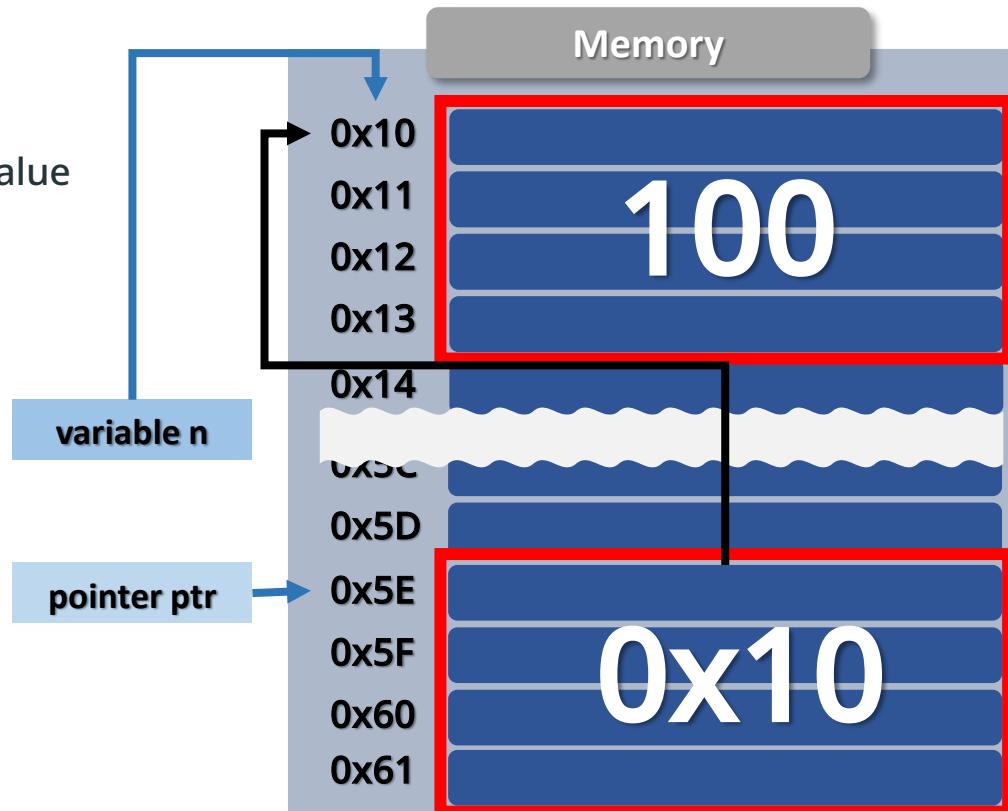
Arrays and pointers

A pointer in C is a variable that stores an address in memory, also known as a pointer variable. Just as a char variable stores a character and an int variable stores an integer, a pointer stores an address value.

- The concept of a pointer
 - The definition of a pointer
 - A variable that stores an address value in memory
 - Referred to as a pointer or pointer variable
 - Use an address operator (&) and a reference operator (*)

int n = 100;

int *ptr = &n;



Source : TCP school homepage

C - array and pointer

Arrays and pointers

A pointer in C is a variable that stores an address in memory, also known as a pointer variable. Just as a char variable stores a character and an int variable stores an integer, a pointer stores an address value.

- Pointer operators
 - Address operator (&)
 - Used before the name of a variable to return the address value of that variable
 - Read as ampersand, also known as the bungie operator
 - Reference operator (*)
 - Used before the name or address of a pointer
 - Used to return the value stored at the address pointed to by the pointer
 - When used as a binary operator, it is used for a multiplication operation; when used to declare a pointer or to access memory, it is used for a multiplication operation.

C - array and pointer

Arrays and pointers

A pointer in C is a variable that stores an address in memory, also known as a pointer variable. Just as a char variable stores a character and an int variable stores an integer, a pointer stores an address value.

- Declaring a pointer

Type ***pointer name;**

- After declaring a pointer and before using a reference operator, you must first initialize it.
 - Prevent unintended changes to memory values
- Initialize a pointer at the same time as its declaration

Type ***pointer name = &variable name;**

or

Type ***pointer name = address value;**

- Pointer reference

- Reference the address value of a pointer together with the data at the address value the pointer is pointing to.

```
int x = 10; // Declare a variable  
int *ptr = &x; // Declare and initialize a pointer  
int *pptr = &ptr; // Reference to a pointer
```

C - array and pointer

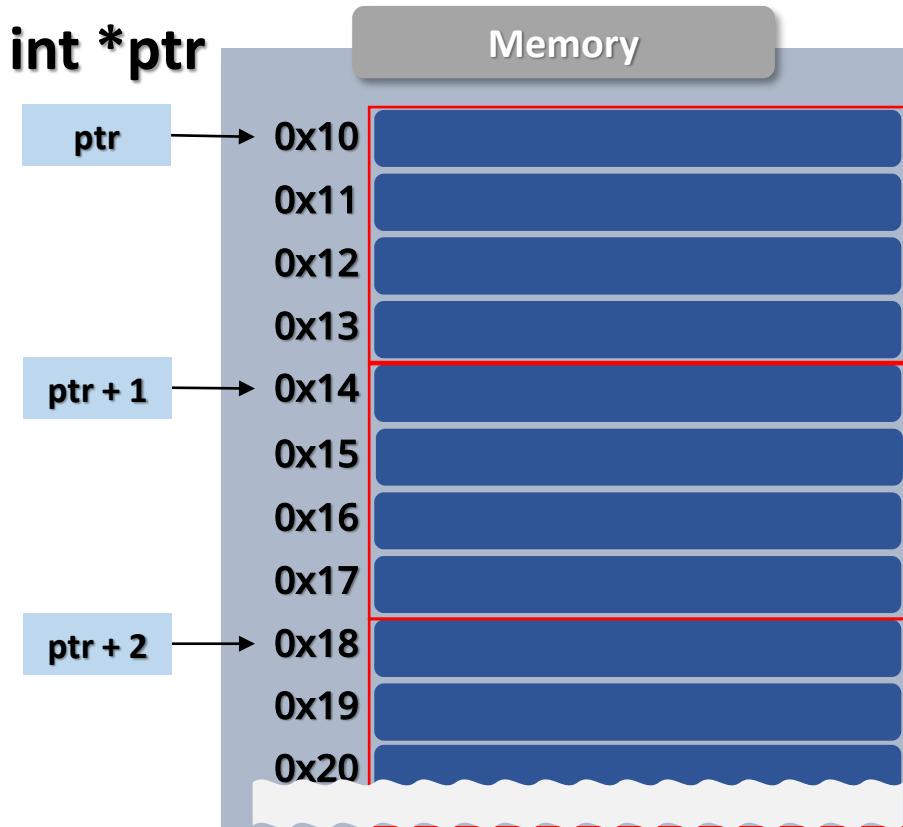
Arrays and pointers

A pointer in C is a variable that stores an address in memory, also known as a pointer variable. Just as a char variable stores a character and an int variable stores an integer, a pointer stores an address value.

- Pointer operations

- Rules

- Addition, multiplication, and division between pointers are not recognized as having no meaning
 - Subtraction between pointers indicates the relative distance between two pointers.
 - You can add or subtract integers to a pointer, but not real numbers.
 - Pointers can be substituted or compared.



Source : TCP school homepage

C - array and pointer

Arrays and pointers

A pointer in C is a variable that stores an address in memory, also known as a pointer variable. Just as a char variable stores a character and an int variable stores an integer, a pointer stores an address value.

- Sample question 1 on pointer operation

- The width of an incremented pointer address after a pointer operation that increments by 1 is equal to the size of the type of the pointer variable.
- Source file - "pointer2.c"

```
#include <stdio.h>

int main(void)
{
    char *ptr_char = 0;
    int *ptr_int = NULL;
    double *ptr_double = 0x00;

    printf("ptr_char current address value %#x\n", ptr_char);
    printf("ptr_int current address value %#x\n", ptr_int);
    printf("ptr_double current address value %#x\n", ptr_double);

    printf("After incrementing ptr_char by 1, the address value is %#x\n", ++ptr_char);
    printf("After incrementing ptr_int by 1, the address value is %#x\n", ++ptr_int);
    printf("After incrementing ptr_double by 1, the address value is %#x\n", ++ptr_double);
    return 0;
}
```

C - array and pointer

Arrays and pointers

- Sample question 2 on pointer operation
 - Source file - "pointer2.c"

```
#include <stdio.h>

int main(void)
{
    int num01 = 10;
    int num02 = 20;
    int *ptr_num01 = &num01;
    int *ptr_num02 = &num02;

    if (ptr_num01 != ptr_num02) // Comparison operation between pointers
    {
        printf("The value stored at the address pointed to by pointer ptr_num01 is %d.\n", *ptr_num01);
        printf("The value stored at the address pointed to by pointer ptr_num02 is %d.\n", *ptr_num02);
        printf("Pointers ptr_num01 and ptr_num02 are currently pointing to different addresses.\n\n");
        ptr_num02 = ptr_num01; // assignment between pointers
    }
    printf("The value stored at the address pointed to by pointer ptr_num01 is %d.\n", *ptr_num01);
    printf("The value stored at the address pointed to by pointer ptr_num02 is %d.\n", *ptr_num02);
    if (ptr_num01 == ptr_num02) // Comparison operation between pointers
    {
        printf("Pointers ptr_num01 and ptr_num02 are currently pointing to the same address.\n");
    }
    return 0;
}
```

C - array and pointer

Arrays and pointers

A pointer in C is a variable that stores an address in memory, also known as a pointer variable. Just as a char variable stores a character and an int variable stores an integer, a pointer stores an address value.

- How to pass arguments
 - Call by value
 - Call by reference
 - In fact, C does not support call by reference
 - A pointer looks like passing an argument by reference, because the address value of its value is passed.
 - In conclusion, C is all about passing by value, and the only difference is whether what is being passed is a value or an address value.

	Call by value	Call by reference	Call by address
How functions are called	Call by value	Call by reference	Call by address
Object for passing arguments	Passing values (constants)	Passing references of arguments	Passing addresses of arguments
Impact on the called function	Not impacted	Impacted	Impacted
Parameter	int x, int y	int &x, int &y	int *x, int *y
If supported or not	Supported	Not supported	Supported

C - array and pointer

Arrays and pointers

A pointer in C is a variable that stores an address in memory, also known as a pointer variable. Just as a char variable stores a character and an int variable stores an integer, a pointer stores an address value.

- Example of the argument passing method
 - Source file - "function_call.c"

```
#include <stdio.h>

void call_by_value(int number)
{
    number = 20; // No impact on the original variable
}
void call_by_addr(int *number)
{
    *number = 30; // Have impact on the original variable
}
int main(void)
{
    int number = 10;
    call_by_value(number);
    printf("%d\n", number);
    call_by_addr(&number);
    printf("%d\n", number);
    return 0;
}
```

C - array and pointer

Arrays and pointers

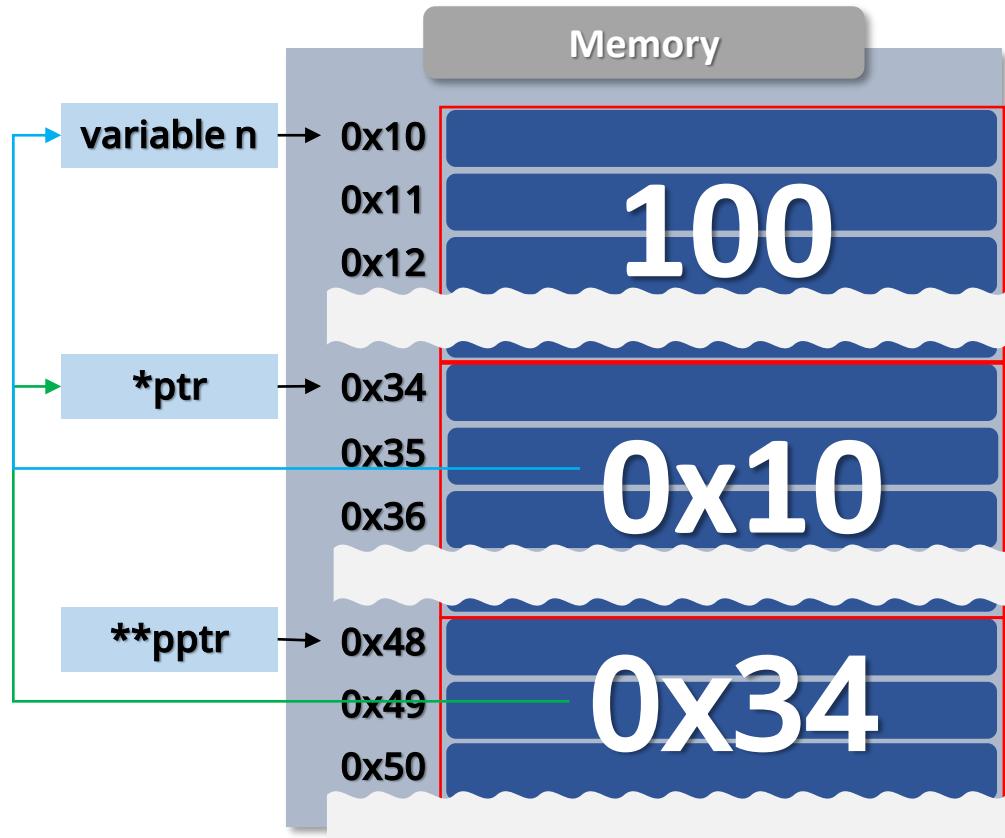
A pointer in C is a variable that stores an address in memory, also known as a pointer variable. Just as a char variable stores a character and an int variable stores an integer, a pointer stores an address value.

- Various pointers
 - Pointer to pointer
 - Refer to a pointer to a variable
 - Use a reference operator (*) twice to express
 - Also known as a double pointer

int n = 100;

int *ptr = &n;

int **pptr = &ptr;



Source : TCP school homepage

C - array and pointer

Arrays and pointers

A pointer in C is a variable that stores an address in memory, also known as a pointer variable. Just as a char variable stores a character and an int variable stores an integer, a pointer stores an address value.

- Various pointers

- Void pointer
 - Pointer that does not specify the data type
 - Can be any value such as variable, function, pointer, etc.
 - Operations such as pointer operations and memory references are not possible
 - A pointer that can only store an address value
 - First convert to the explicit type you want to use, then use it

```
<Example>
int num = 10;
void *ptr_num = &num;

printf("The value being stored by variable num is %d.\n", num);
printf("The value stored at the address pointed to by void pointer ptr_num is %d.\n", *(int *)ptr_num);

*(int *)ptr_num = 20;
printf("The value stored at the address pointed to by void pointer ptr_num is %d.\n", *(int *)ptr_num);
```

C - array and pointer

Arrays and pointers

A pointer in C is a variable that stores an address in memory, also known as a pointer variable. Just as a char variable stores a character and an int variable stores an integer, a pointer stores an address value.

- Various pointers
 - Function pointer
 - The name of the function is a pointer constant that points to the starting address of the function.
 - A pointer constant that points to the starting address of a function
 - The pointer type is determined by the function's return value and parameters.
 - You must know the function's prototype to create a function pointer appropriate for that function.

```
<function prototype>
void func(int, int);
```

```
<function pointer>
void (*ptr_func) (int, int);
```

C - array and pointer

Arrays and pointers

- Various pointers
 - Example of the function pointer (1/3)

```
#include <stdio.h>

double add(double, double);
double sub(double, double);
double mul(double, double);
double div(double, double);
double calculator(double , double, double (*func)(double, double));

int main(void)
{
    double (*calc)(double, double) = NULL; // Declare a function pointer
    double result = 0;

    double num01 = 3, num02 = 5;
    char oper = '*';

    switch (oper)
    {
        case '+':
            calc = add;
            break;
        case '-':
            calc = sub;
            break;
```

C - array and pointer

Arrays and pointers

- Various pointers
 - Example of the function pointer (2/3)

```
case '*':  
    calc = mul;  
    break;  
case '/':  
    calc = div;  
    break;  
default:  
    puts("Only arithmetic operations (+, -, *, /) are supported.");  
}  
  
result = calculator(num01, num02, calc);  
printf("The result of the arithmetic operation is %lf.\n", result);  
return 0;  
}  
  
double add(double num01, double num02)  
{  
    return num01 + num02;  
}  
  
double sub(double num01, double num02)  
{  
    return num01 - num02;  
}
```

C - array and pointer

Arrays and pointers

- Various pointers
 - Example of the function pointer (3/3)

```
double mul(double num01, double num02)
{
    return num01 * num02;
}

double div(double num01, double num02)
{
    return num01 / num02;
}

double calculator(double num01, double num02, double (*func)(double, double))
{
    return func(num01, num02);
}
```

C - array and pointer

Arrays and pointers

Pointers and arrays have a very close relationship, and in some ways they can be used interchangeably. The name of an array is the same as a pointer, except that its value is an immutable constant. Thus, the name of an array is a pointer constant.

- Relationship between arrays and pointers
 - Possible to replace the name of an array with a pointer and use the pointer as the name of the array
 - When calculating the size of an array, there is a difference between the name of the array and the pointer.

```
#include <stdio.h>

int main(void)
{
    int arr[3] = {10, 20, 30};
    int *ptr = arr;

    printf(" Access the elements of an array by the array name : %d %d %d\n", arr[0], arr[1], arr[2]);
    printf("Access the elements of an array using pointers : %d %d %d\n", ptr[0], ptr[1], ptr[2]);
    printf(" Calculate the size of an array by its name : %d\n", sizeof(arr));
    printf("Calculate the size of an array using pointers: %d\n", sizeof(ptr));

    return 0;
}
```

C - array and pointer

Arrays and pointers

Pointers and arrays have a very close relationship, and in some ways they can be used interchangeably. The name of an array is the same as a pointer, except that its value is an immutable constant. Thus, the name of an array is a pointer constant.

- Relationship between arrays and pointers
 - Pointer operations on arrays
 - When arr is the name of an array or a pointer to it and n is an integer, $\text{arr}[n] == *(\text{arr} + n)$

```
#include <stdio.h>

int main(void)
{
    int arr[3] = {10, 20, 30};

    printf("Access the elements of an array by the array name : %d %d %d\n", arr[0], arr[1], arr[2]);
    printf("Access the elements of an array by pointer operation on the array name : %d %d %d\n", *(arr+0), *(arr+1), *(arr+2));

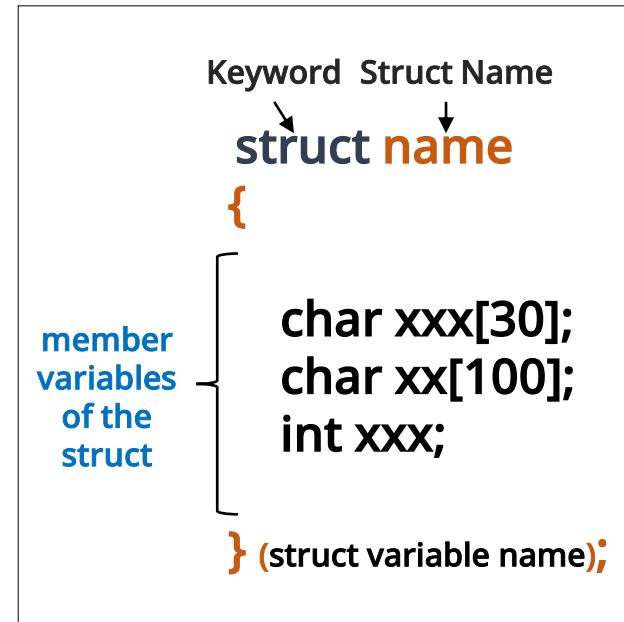
    return 0;
}
```

C - structure

Structures

A structure (struct) is a user-defined type created from the C language's default type. It can express complex data that cannot be expressed by basic types alone. If an array is a set of variables of the same type, a structure is a set of variables of different types expressed in a single type.

- How to define and declare a struct
 - Use a struct keyword
 - Each variable is called a member or member variable.
 - A defined struct is also known as a user-defined data type
 - Declare a struct variable
 - struct [struct name] [struct variable name].
 - E.g., struct name name_st
 - A struct variable name can be specified when the struct is defined.



C - structure

Structures

A structure (struct) is a user-defined type created from the C language's default type. It can express complex data that cannot be expressed by basic types alone. If an array is a set of variables of the same type, a structure is a set of variables of different types expressed in a single type.

- Why do we use structs?
 - Use multiple types of variables to effectively process data
 - E.g., 40 variable declarations to store the names, ages, phone numbers, and addresses of 10 people
 - → Only 10 struct declarations named private data needed for the same data

```
char name1[30];
int age1;
char call_num1[15];
char address1[40];

.
.
.
.
.
.

char name10[30];
int age10;
char call_num10[15];
char address10[40];
```

```
struct private_data
{
    char name[30];
    int age;
    char call_num[15];
    char address[40];
}

struct private_data p1;
struct private_data p2;

.
.
.

struct private data p10;
```

C - structure

Structures

A structure (struct) is a user-defined type created from the C language's default type. It can express complex data that cannot be expressed by basic types alone. If an array is a set of variables of the same type, a structure is a set of variables of different types expressed in a single type.

- **typedef keyword**
 - Used to give a new name to an already existing type
 - Using the **typedef** keyword to declare a new structure name avoids using the **struct** keyword every time

How to use

```
typedef struct [struct name] [new structure name];
```

Can be used concurrently with a **struct definition**, in which case the **struct name** can be

```
typedef struct (struct name)
{
    MemberVariableType1 MemberVariableName1;
    MemberVariableType2 MemberVariableName2;
    ...
} A new name for the structure;
```

```
typedef struct
{
    char title[30];
    char author[30];
    int price;
} TEXTBOOK;
```

C - structure

Structures

A structure (struct) is a user-defined type created from the C language's default type. It can express complex data that cannot be expressed by basic types alone. If an array is a set of variables of the same type, a structure is a set of variables of different types expressed in a single type.

- How to access struct members
 - Use the member operator (.)

[struct variable name].[member variable name].

- The address value of a structure and the address value of the first member variable of the structure are always the same.

- How to initialize struct variables
 - Use member operators (.) and curly braces ({})

- How to initialize only desired member variables (uninitialized member variables are initialized to 0)

[struct variable name] = { .member variable1 name = initial value, .member variable2 name = initial value, ... }

- How to initialize member variables in the same order as an array.

[struct variable name] = { initial value of .member variable1, initial value of .member variable2, ... }

A structure (struct) is a user-defined type created from the C language's default type. It can express complex data that cannot be expressed by basic types alone. If an array is a set of variables of the same type, a structure is a set of variables of different types expressed in a single type.

- Example of the structs

```
#include <stdio.h>

typedef struct
{
    char title[30];
    char author[30];
    int price;
} BOOK;

int main(void)
{
    BOOK my_book = {"Python", "ACS", 28000};
    BOOK c_book = {.title = "C language", .price = 30000};

    printf("First book title : %s, Author : %s, Price : %d yuan\n", my_book.title, my_book.author, my_book.price);
    printf("Second book title : %s, Author : %s, Price : %d yuan\n", c_book.title, c_book.author, c_book.price);

    return 0;
}
```

C - structure

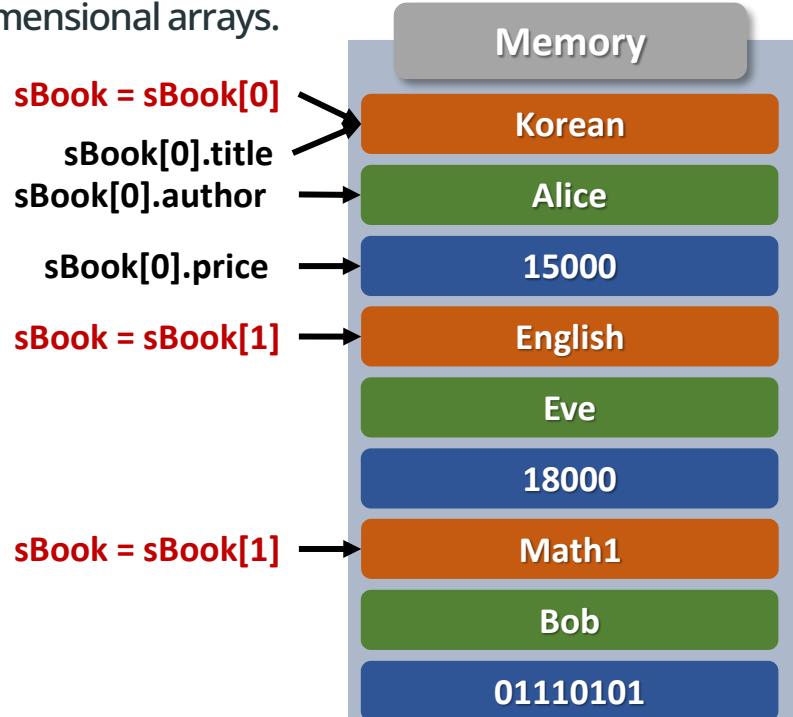
Structures

A structure (struct) is a user-defined type created from the C language's default type. It can express complex data that cannot be expressed by basic types alone. If an array is a set of variables of the same type, a structure is a set of variables of different types expressed in a single type.

- How to declare an array of structs

- Even structs can be created as arrays with no restrictions on the types that can be arrayed.
- Struct arrays are initialized in the same way as two-dimensional arrays.

```
#include <stdio.h>
struct book
{
    char title[30];
    char author[30];
    int price;
};
int main(void)
{
    struct book sBook[3] = {
        {"Korean", "Hong Gil-dong", 15000},
        {"English", "Yi Sun-sin", 18000},
        {"math1", "Kang Kam-chan", 10000}
    };
    printf("%s, %s, %s\n",
sBook[0].title, sBook[1].title, sBook[2].title);
    return 0;
}
```



C - structure

Structures

A structure (struct) is a user-defined type created from the C language's default type. It can express complex data that cannot be expressed by basic types alone. If an array is a set of variables of the same type, a structure is a set of variables of different types expressed in a single type.

- Struct pointer
 - How to declare a struct pointer

```
struct [struct name] *[struct pointer name];
```

- Use the address operator (&) when assigning to a struct pointer
- How to access struct members from a struct pointer
 - Use the reference operator (*)
 - Reference operators (*) have lower operator precedence than member operators (.) and must be enclosed in square brackets ([]).

```
(* [struct pointer name]).[member variable name].
```

- Use the arrow operator (->)
 - Commonly used methods

```
[struct pointer name] -> [member variable name].
```

A structure (struct) is a user-defined type created from the C language's default type. It can express complex data that cannot be expressed by basic types alone. If an array is a set of variables of the same type, a structure is a set of variables of different types expressed in a single type.

- Struct pointer example

```
#include <stdio.h>
#include <string.h>

typedef struct {
    char title[30];
    char author[30];
    int price;
} book;
int main(void)
{
    book my_book = {"Complete Anatomy of the C Language", "Hong Gil-dong", 35000};
    book* ptr_book;

    ptr_book = &my_book;
    strcpy((*ptr_book).title, "C Language Master");
    strcpy(ptr_book ->author, "Yi Sun-sin");
    my_book.price = 32000;

    printf("Book title : %s, Author : %s, Price : %d\n", my_book.title, my_book.author, my_book.price);
    return 0;
}
```

A structure (struct) is a user-defined type created from the C language's default type. It can express complex data that cannot be expressed by basic types alone. If an array is a set of variables of the same type, a structure is a set of variables of different types expressed in a single type.

- Functions and structures

- Structures can be used as arguments passed to functions or as return values.
- Example of passing the struct member variable function argument

```
#include <stdio.h>
typedef struct {
    int savings;
    int loan;
} PROP;
int calcProperty(int, int);
int main(void) {
    int hong_prop;
    PROP hong = {10000000, 4000000};
    hong_prop = calcProperty(hong.savings, hong.loan);
    printf("Hong Gil-dong's assets - total %d KRW excluding savings %d KRW and loans %d KRW\n", hong.savings, hong.loan, hong_prop);
    return 0;
}
int calcProperty(int s, int l)
{
    return (s - l);
}
```

A structure (struct) is a user-defined type created from the C language's default type. It can express complex data that cannot be expressed by basic types alone. If an array is a set of variables of the same type, a structure is a set of variables of different types expressed in a single type.

- Functions and structures
 - Example of passing the struct address function argument

```
#include <stdio.h>
typedef struct {
    int savings;
    int loan;
} PROP;
int calcProperty(PROP *);
int main(void) {
    int hong_prop;
    PROP hong = {10000000, 4000000};
    hong_prop = calcProperty(&hong);
    printf("Hong Gil-dong's assets - total %d KRW excluding savings %d KRW and loans %d KRW\n", hong.savings, hong.loan, hong_prop);
    return 0;
}

int calcProperty(PROP *money)
{
    money->savings = 100;
    return (money->savings - money->loan);
}
```

A structure (struct) is a user-defined type created from the C language's default type. It can express complex data that cannot be expressed by basic types alone. If an array is a set of variables of the same type, a structure is a set of variables of different types expressed in a single type.

- Nested structures

- When defining a struct, you can include another struct as a member variable

```
#include <stdio.h>

struct name {
    char first[30];
    char last[30];
};

struct friends {
    struct name friend_name;
    char address[30];
    char job[30];
};

int main(void) {
    struct friends hong = { { "Gil-dong", "Hong" }, "Yeoksam-dong, Gangnam-gu, Seoul", "student" };
    printf("%s\n\n", hong.address);
    printf("To %s%s,\n", hong.friend_name.last, hong.friend_name.first);
    printf("How have you been? Are you still %s?\n", hong.job);
    return 0;
}
```

C - structure

Structures

- Practice question

- Source file - "struct1.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

struct score {
    char name[20];
    int score;
};

struct score score[5];
void ask() {
    for (int i = 0; i < 3; i++) {
        printf("Please enter the student name : ");
        scanf("%s", _____);
        printf("Please enter your score : ");
        scanf("%d", _____);
    }
    printf("----- [Grade Status] ----- \n");
    for (int i = 0; i < 3; i++) {
        printf("\nName : %s\n", _____);
        printf("Score : %d\n", _____);
    }
}
void main() {
    ask();
}
```

C - structure

Structures

- Answer to the practice question
 - Source file - "struct1.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

struct score {
    char name[20];
    int score;
};

struct score score[5];
void ask() {
    for (int i = 0; i < 3; i++) {
        printf("Please enter the student name : ");
        scanf("%s", &score[i].name);
        printf("Please enter your score : ");
        scanf("%d", &score[i].score);
    }
    printf("----- [Grade Status] ----- \n");
    for (int i = 0; i < 3; i++) {
        printf("\nName : %s\n", score[i].name);
        printf("Score : %d\n", score[i].score);
    }
}
void main() {
    ask();
}
```

C - strings

String-related functions

C supports a variety of functions for handling strings, including comparison, length, and copy, and these string handling functions are packaged in the "string.h" library.

- String overview

- Strings can be stored using arrays
- Strings ends with NULL ('\0').
- Input and output strings using the format specifier "%s"

```
#include <stdio.h>

int main(void)
{
    char string1[20] = "hello world";
    char string2[20] = { 'H','E','L','L','O','\n',
    ', 'W','O','R','L','D','\0', 'a', 'b', 'c' };

    // Array length exceeded
    // char string1[10] = "hello world";

    // Declaration and initialization must occur at the same time.
    // char string2[20];
    // string2 = "HELLO WORLD";

    printf("%s\n", string1);
    printf("%s\n", string2);

    return 0;
}
```

C supports a variety of functions for handling strings, including comparison, length, and copy, and these string handling functions are packaged in the "string.h" library.

- String comparison functions

- strcmp function

- strcmp(["string1" or array], ["string2" or array]);
 - Vulnerable function because the length of the string to be compared is not specified

- strncmp function

- strncmp(["string1" or array], ["string2" or array], [length to compare]);
 - Safe function because the length of the strings to be compared is specified

Semantics (vs. ASCII code)	Return value	
	strcmp	strncmp
String1 is greater than string2	Positive numbers	Positive numbers
String1 is equal to string2	0	0
String1 is less than string2	Negative numbers	Negative numbers

C supports a variety of functions for handling strings, including comparison, length, and copy, and these string handling functions are packaged in the "string.h" library.

- Example of the string comparison function
 - Source file - "string_comparison.c"

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char string1[20] = "hello world";
    char string2[20] = "hello everyone";

    if (strcmp(string1, string2) == 0)
        printf("string1 and string2 match\n");
    else
        printf("string1 and string2 do not match\n");

    if (strncmp(string1, string2, 5) == 0) // Compare hello to hello
        printf("string1 and string2 match\n");
    else
        printf("string1 and string2 do not match\n");
    return 0;
}
```

C supports a variety of functions for handling strings, including comparison, length, and copy, and these string handling functions are packaged in the "string.h" library.

- String copy functions

- strcpy function

- strcpy([array(location to store)], ["string2" or array(string to store)]);
 - Vulnerable function because the length of the string to be copied is not specified
 - Return the memory address of the stored location

- strncpy function

- strncpy([array(location to store)], ["string2" or array(string to store)], [length to copy]);
 - Safe function because the length of the strings to compare is specified
 - Return the memory address of the stored location

C supports a variety of functions for handling strings, including comparison, length, and copy, and these string handling functions are packaged in the "string.h" library.

- Example of the string copy function
 - Source file - "string_copy.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

int main(void)
{
    char string1[20] = { 0, };
    char string2[20] = "hello world";
    char string3[20] = { 0, };

    strcpy(string1, string2);
    printf("%s\n", string1);

    strncpy(string3, string2, 4);
    printf("%s\n", string3);

    return 0;
}
```

C supports a variety of functions for handling strings, including comparison, length, and copy, and these string handling functions are packaged in the "string.h" library.

- String length checking functions
 - `strlen` function
 - `strlen(["string" or array]);`
 - Return the length of a string
- Example of the string length checking function
 - Source file - "string_length.c"

```
#include<stdio.h>
#include<string.h>

int main(void)
{
    char string1[20] = "hello world";

    printf("%d\n", strlen(string1));

    return 0;
}
```

C supports a variety of functions for handling strings, including comparison, length, and copy, and these string handling functions are packaged in the "string.h" library.

- String concatenation functions
 - strcat function
 - strcat([array(where to concat.)], ["string" or array(where to concat.)]);
 - Vulnerable function because the length of the string to concatenate is not specified
 - strncat function
 - strncat([array(where to concat.)], ["string" or array(where to concat.)], [length to concat.]);
 - Safe function because the length of the string to concatenate is specified

C supports a variety of functions for handling strings, including comparison, length, and copy, and these string handling functions are packaged in the "string.h" library.

- Example of the string concatenation function
 - Source file - "string_concatenate.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

int main(void)
{
    char string1[20] = "hello";
    char string2[20] = "world";

    strcat(string1, string2);
    printf("%s\n", string1);

    strncat(string1, string2, 3);
    printf("%s\n", string1);

    return 0;
}
```

C supports a variety of functions for handling strings, including comparison, length, and copy, and these string handling functions are packaged in the "string.h" library.

- Other string processing functions
 - strchr function
 - Search for characters within a search target string
 - strchr([array (what to search for)], ["character" or variable (the character to search for)]);
 - Return the memory address of a matching character if the search is successful
 - strstr function
 - Search for a string within a search target string
 - strstr([array(object to search for)], ["string" or array(string to search for)]);
 - Return the memory address of the starting point of the string if the search for a matching string is successful

C supports a variety of functions for handling strings, including comparison, length, and copy, and these string handling functions are packaged in the "string.h" library.

- Other string processing functions
 - strtok function
 - Function to truncate a string based on a specific character
 - strtok(["string" or array (characters to truncate)], ["character" or variable (characters to truncate)]);
 - If the string contains "NULL", it means the trailing string left after truncation.
 - Equivalent to truncating a string with the strtok function once more after the strtok function is executed.

C supports a variety of functions for handling strings, including comparison, length, and copy, and these string handling functions are packaged in the "string.h" library.

- Example 1 of the other string processing function
 - Source file - "string_search.c"

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    char string1[20] = "hello world";
    char string2[20] = "hello C language";

    printf("%s\n", strchr(string1, 'o'));
    printf("%s\n", strstr(string2, "lo"));

    return 0;
}
```

C supports a variety of functions for handling strings, including comparison, length, and copy, and these string handling functions are packaged in the "string.h" library.

- Example 2 of the other string processing function
 - Source file - "string_slice.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <string.h>

int main(void)
{
    char string1[20] = "hello C language";

    printf("%s\n", strtok(string1, " "));
    printf("%s\n", strtok(NULL, " "));
    printf("%s\n", strtok(NULL, " "));

    return 0;
}
```

C - file input/output

File input and output

In order to input and output data from a file, we need to create a separate stream in addition to the standard stream.

- File input/output overview
 - Need to create a separate stream in addition to the standard stream to perform file input/output
 - Use file descriptors and streams
 - File descriptors
 - Low-level file input and output
 - Assign numbers to files to organize them
 - The operating system itself assigns file descriptors.
 - File streams
 - High-level file input and output
 - File structures are used to get information from a file, allowing for input/output to and from a variety of devices.
 - File streams include file descriptors

C - file input/output

File input and output

In order to input and output data from a file, we need to create a separate stream in addition to the standard stream.

- Create a file stream
 - fopen() function
 - Create a new file or load an existing file
 - fopen([filename], [generation mode])
 - Return a file stream
 - ▶ fclose() function
 - Close the imported file
 - fclose([file pointer])

Generation mode	Description	If the file does not exist
r	Read-only mode. Read the contents of the file if it exists	Unable to execute
w	Write-only mode. If the file exists, delete the existing contents and create new one	Create a file
a	Append-only mode. If the file exists, append it after the existing content	Create a file
r+	Read & write mode. Read the contents of a file if it exists	Unable to execute
w+	Read & write mode. If the file exists, delete the existing contents and create new one	Create a file
a+	Read & append mode. If the file exists, append after the existing content	Create a file

C - file input/output

File input and output

In order to input and output data from a file, we need to create a separate stream in addition to the standard stream.

- File input and output functions
 - File output function : output to a file not to the console window
 - fputc([character or variable],[file pointer])
 - fputs([array],[file pointer])
 - fprintf([file pointer][format or string],[string or array])
 - File input function : take in data from a file and save it
 - fgetc([file pointer])
 - fgets([array],[length to receive input],[file pointer])
 - fscanf([file pointer],[format specifier],[variable or array])

C - file input/output

File input and output

- Example 1 and 2 of the file input and output

- Source file - "fileIO1.c"

```
// Create a stream in file read mode
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    FILE *fp = fopen("data.txt", "r");

    if(fp)
        printf("File open(read) success!\n");
    else
        printf("Error!\n");

    fclose(fp);
    return 0;
}
```

- Source file - "fileIO2.c"

```
// Create a stream in file write mode
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    FILE *fp = fopen("data.txt", "w");

    if(fp)
        printf("File open(write) success!\n");
    else
        printf("Error!\n");

    fclose(fp);
    return 0;
}
```

C - file input/output

File input and output

- Example 3 of the file input and output
 - Source file - "fileIO3.c"

```
// Create a stream in file read mode
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(void)
{
    FILE *fp = fopen("data.txt", "r");
    char buffer[30] = "";

    if (fp)
        printf("File open(read) success!\n");
    else
        printf("Error!\n");

    fgets(buffer,10,fp); // File input function
    printf("%s\n", buffer);

    fprintf(fp, "\n%s", buffer); // File output function
    fclose(fp);

    return 0;
}
```

C - memory management

Memory management and dynamic allocation

C provides functions that allow you to allocate memory flexibly, so that you do not run out of memory or have more memory than expected when programming.

- Dynamic allocation of memory overview
 - Allocate memory when process runs -> stay in memory until process ends
 - Freedom to allocate and deallocate memory -> Return unnecessary resident memory via return
 - Different memory regions are used when allocating memory statically vs. dynamically
 - Memory region on static allocation : stack
 - Memory region on dynamic allocation : heap
 - Defined in the stdlib.h header file

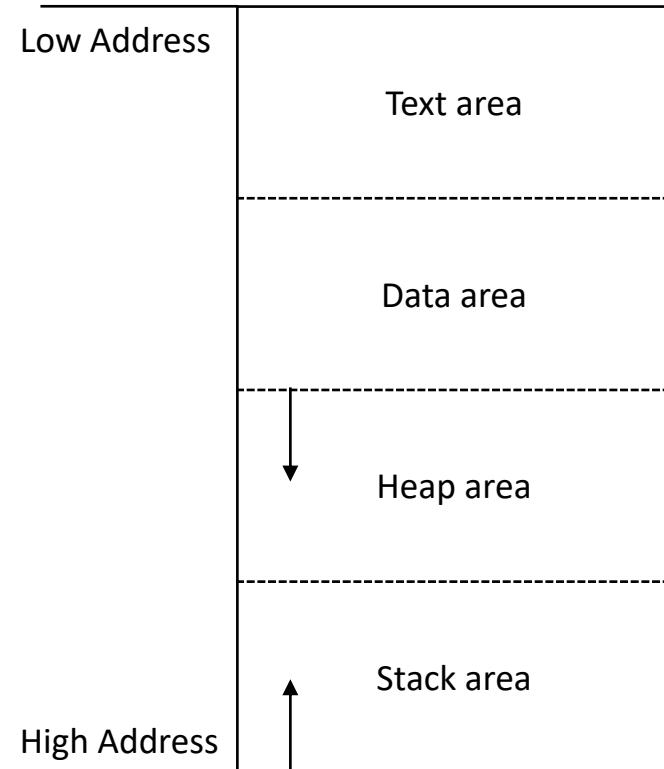
C - memory management

Memory management and dynamic allocation

C provides functions that allow you to allocate memory flexibly, so that you do not run out of memory or have more memory than expected when programming.

- Memory Structure

- Text area
 - The space of the code that makes up a program
- Data area
 - The space where global variables are stored
 - Assign until the end of the program
- Heap area
 - The space saved when allocating memory dynamically
 - Add data upwards if there are additional allocations
- Stack Area
 - Areas where variables and parameters are assigned
 - Add data downward if there are additional allocations



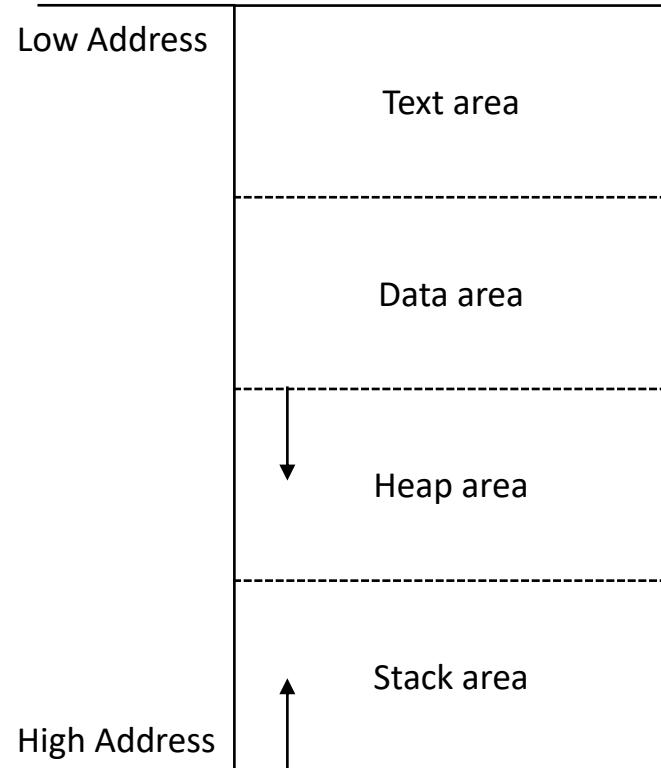
C - memory management

Memory management and dynamic allocation

C provides functions that allow you to allocate memory flexibly, so that you do not run out of memory or have more memory than expected when programming.

- Heap

- Occupy the largest area of memory
- The area used when allocating memory dynamically
- Store data from the lower to upper address direction
- Memory must be returned after use
- Implemented as a linked list
 - Implemented as a linked list of memory regions called chunks.



C - memory management

Memory management and dynamic allocation

C provides functions that allow you to allocate memory flexibly, so that you do not run out of memory or have more memory than expected when programming.

- Functions used to dynamically allocate memory

- `malloc([size to allocate])`
 - Dynamically allocate any size that needs allocation
 - `malloc` function return value : void pointer
 - Must be converted to the desired data type
- `calloc([no. of memory to allocate],[size to allocate])`
 - Allocate multiple dynamic memory areas at once
 - Automatically reset to 0 on allocation
- `realloc([memory address to replace, [size to replace]])`
 - Change the size of an already allocated memory area
- `free([allocated memory address])`
 - Return (free) an allocated memory area

```
<malloc example>
// Need to specify data type before malloc function
// Dynamically allocate memory by the int type size
int *x;
x = (int*) malloc (sizeof(int));

<calloc example>
// Dynamically allocate 5 memory units by the int type size
int *x2;
x2 = (int*) calloc(5,sizeof(int));

<realloc example>
// Dynamically allocate 4 bytes and reallocate 80 bytes
int *x;
x = (int*) malloc (sizeof(int));
realloc(x, sizeof(int)*20);

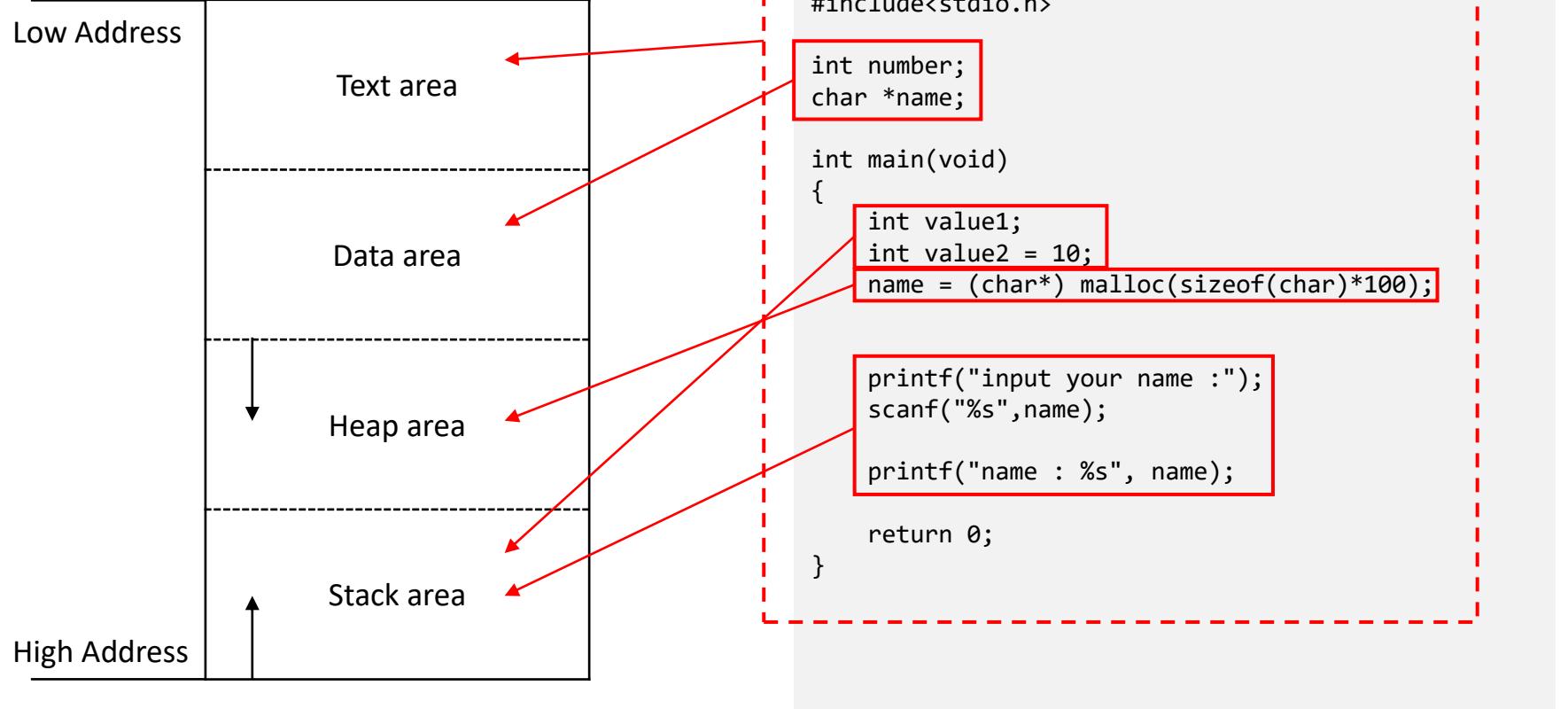
<free example>
// Free dynamic memory allocated for x
int *x;
x = (int*) malloc (sizeof(int));
free(x);
```

C - memory management

Memory management and dynamic allocation

C provides functions that allow you to allocate memory flexibly, so that you do not run out of memory or have more memory than expected when programming.

● C - memory management



C - memory management

Memory management and dynamic allocation

- Example of the memory dynamic allocation
 - Source file - "dynamic1.c"

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int arr_1[5];
    int *arr_2;
    int i;

    for(i = 0; i < 5; i++) {
        arr_1[i] = i+1;
    }

    arr_2 = (int*) malloc(sizeof(int)*5);
    for(i = 0; i < 5; i++) {
        arr_2[i] = arr_1[i];
        printf("%d ", arr_2[i]);
    }

    free(arr_2);
    return 0;
}
```

C - memory management

Memory management and dynamic allocation

- Example of the memory dynamic allocation
 - Source file - "dynamic2.c"

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int arr_1[10];
    int *arr_2;
    int i;

    for (i = 0; i < 10; i++) {
        arr_1[i] = i + 1;
    }
    arr_2 = (int*)malloc(sizeof(int) * 5);
    for (i = 0; i < 5; i++) {
        arr_2[i] = arr_1[i];
        printf("%d ", arr_2[i]);
    }
    printf("\n");
    realloc(arr_2, sizeof(int) * 10);
    for (i = 0; i < 10; i++) {
        arr_2[i] = arr_1[i];
        printf("%d ", arr_2[i]);
    }
    free(arr_2);
    return 0;
}
```

C - preprocessors

Headers and preprocessing directives

A preprocessing directives (preprocessor) is a tool that processes something before the compilation process. It has many uses, such as substituting integer values, which is often used in macros and code to determine whether to execute based on a simple condition.

- Preprocessing directives overview
 - Starts with a hash (#) and no semicolon (;)
 - Write at the top of the code
 - Types of preprocessing directives
 - File handling directive: #include
 - Shape definition directives:
#define, #undef, etc.
 - Conditional processing directives:
#if, #ifdef, #else, etc.

```
#include <stdio.h> // File handling preprocessor
#define PIE 3.14 // Type definition preprocessor
#define Loop 10

int main(void)
{
    int i, j;
    printf("%f\n", PIE);

    for(i=0;i<Loop;i++)
        printf("%dth iteration\n",i);

    for(j=0;j<Loop;j++)
        printf("%dth iteration\n",j);

    return 0;
}
```

C - preprocessors

Headers and preprocessing directives

- Example of the header - (1/2)

- Source file - "header.c"

```
#define _CRT_SECURE_NO_WARNINGS
#include "header.h"

int main(void)
{
    int user, com;
    char command;

    srand(time(NULL));

    while (1)
    {
        printf("Do you want to roll a dice?");
        scanf("%c", &command);
        if (command == 'y')
        {
            user = (rand() % 6) + 1;
            com = (rand() % 6) + 1;
            printf("User : %d, Computer : %d \n", user, com);
            if (user == com)
                printf("We've drawn even\n");
            else if (user > com)
                printf("You've won\n");
        }
    }
}
```

C - preprocessors

Headers and preprocessing directives

- Example of the header - (2/2)

- Source file - "header.c"

```
else
    printf("You've lost\n");
}
else if (command == 'n')
    return 0;
else
    printf("Please select again\n");
while (getchar() != '\n');
}
```

- Header file - "header.h"

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

C - preprocessors

Headers and preprocessing directives

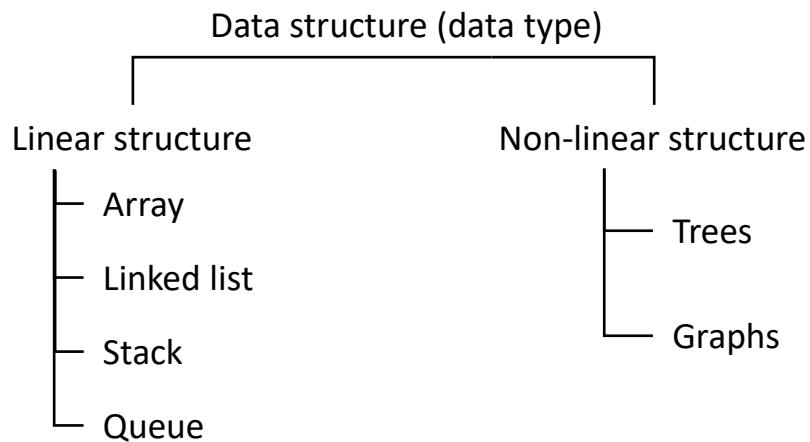
- Practice question
 - Source file - "calc_header.c"
 - Create a calculator program
 - Implement a user-defined function to perform arithmetic operations.
 - User-defined function declarations are implemented in the header file "arithmetic.h" .

C - data structure

Data Structure

A data structure is a way of collecting and organizing data so that it can be stored and used effectively. When it comes to data structures, there are two different types: linear and non-linear. The former links data in a sequential manner, while the latter links data in a contiguous manner.

- Data structures overview
 - The way to efficiently organize and use data in computer memory
 - Used by most programs or software systems
 - Contribute to efficient algorithm design and code

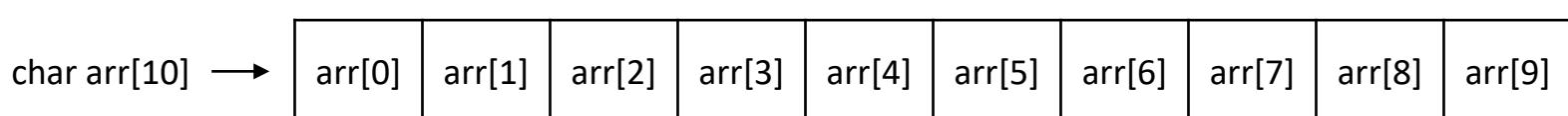


C - data structure

Data Structure

A data structure is a way of collecting and organizing data so that it can be stored and used effectively. When it comes to data structures, there are two different types: linear and non-linear. The former links data in a sequential manner, while the latter links data in a contiguous manner.

- Linear structure types - array
 - The most basic form of data structure
 - Memory space is continuously arranged
 - Very fast to access because users can use indexes to access it
 - Fixed memory size → inefficient when inserting/deleting data



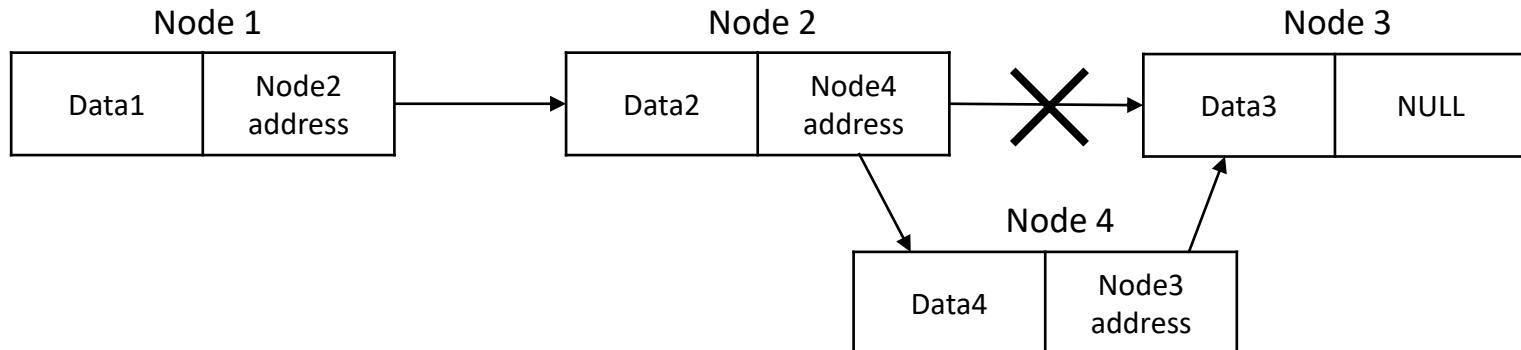
C - data structure

Data Structure

A data structure is a way of collecting and organizing data so that it can be stored and used effectively. When it comes to data structures, there are two different types: linear and non-linear. The former links data in a sequential manner, while the latter links data in a contiguous manner.

- Linear data structure types - linked list

- The way to store data in such a way that nodes are connected by a single line of data and a pointer to it.
- The pointer of one node points to the data address of the next node.
- Easy when inserting/deleting data
- Memory space does not have to be continuous.
- If a node is disconnected, it's hard to know where the next node is.



C - data structure

Data Structure

A data structure is a way of collecting and organizing data so that it can be stored and used effectively. When it comes to data structures, there are two different types: linear and non-linear. The former links data in a sequential manner, while the latter links data in a contiguous manner.

- Linear data structure types - linked list

- Types of linked lists

- Singly linked lists

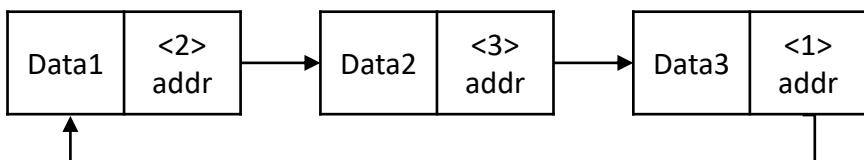
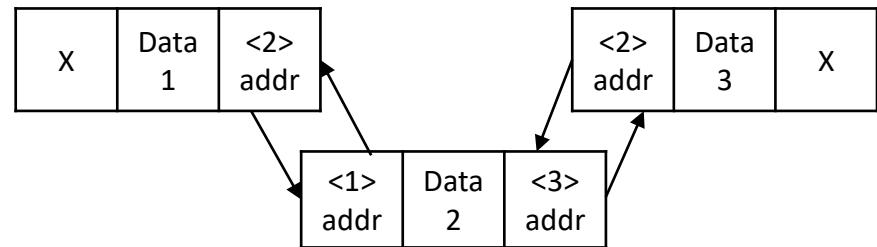
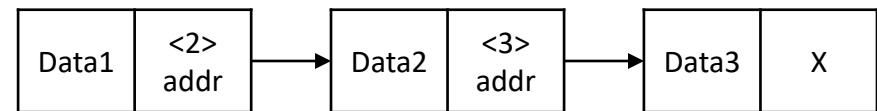
- Use a single pointer to point to the address of the next node

- Doubly linked list

- Use two pointers to point to the previous and next node addresses

- Circular linked lists

- Circular form, where the last node address points to the first node address

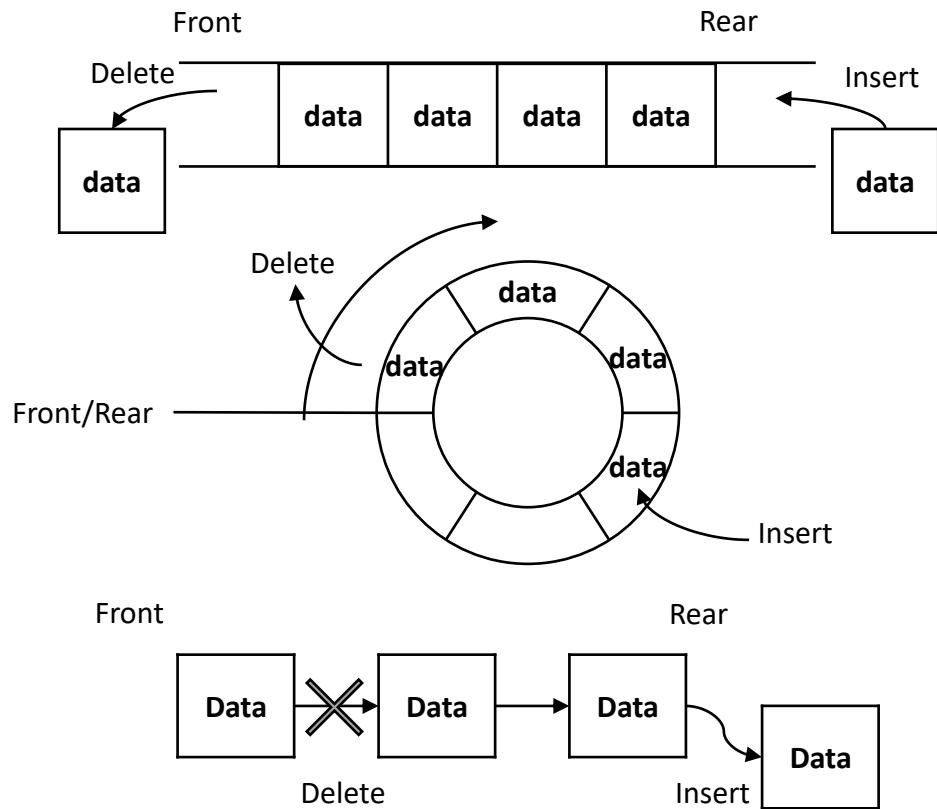


C - data structure

Data Structure

A data structure is a way of collecting and organizing data so that it can be stored and used effectively. When it comes to data structures, there are two different types: linear and non-linear. The former links data in a sequential manner, while the latter links data in a contiguous manner.

- Linear data structure types - queue
 - First In First Out (FIFO) data structures (or Last In, Last Out (LILO) structures)
 - Types of queues
 - Linear queue : rectangular-shaped
 - Overflow occurs even if empty memory exists in the queue
 - Circular queue : circle-shaped
 - Overcoming the disadvantages of linear queues
 - Limited memory size
 - Queuing with connection lists
 - Unlimited memory size



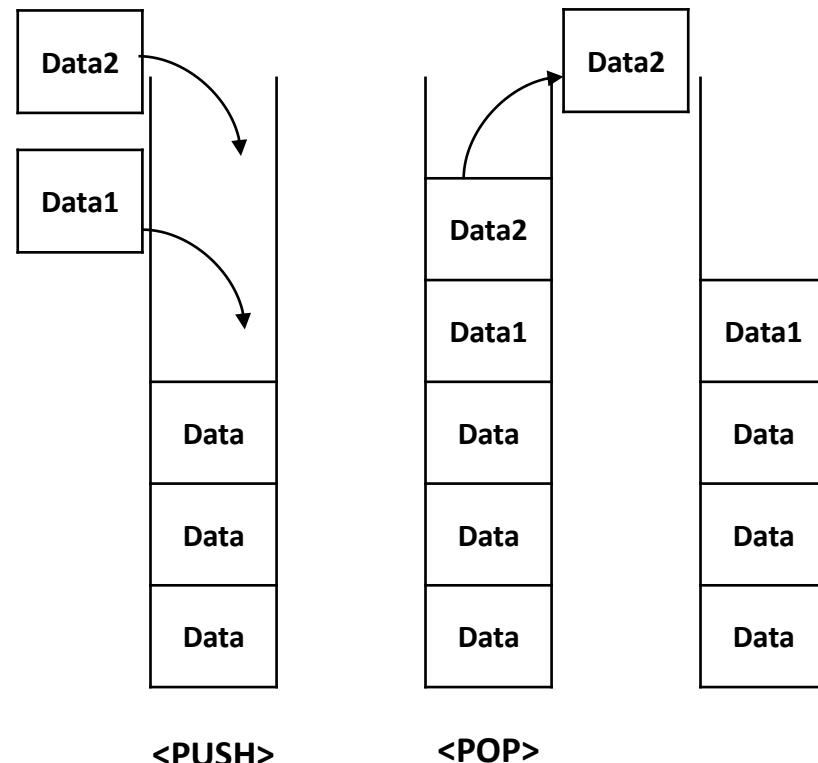
C - data structure

Data Structure

A data structure is a way of collecting and organizing data so that it can be stored and used effectively. When it comes to data structures, there are two different types: linear and non-linear. The former links data in a sequential manner, while the latter links data in a contiguous manner.

- Linear data structure types - stack

- Last In First Out (LIFO) data structure (or First In Last Out (FILO) structure)
- Push operation : add data to the top of the stack
- Pop operation : remove data from the top of the stack
- Stack types
 - Stacks with arrays
 - Stacks with linked lists
 - Unlimited memory size

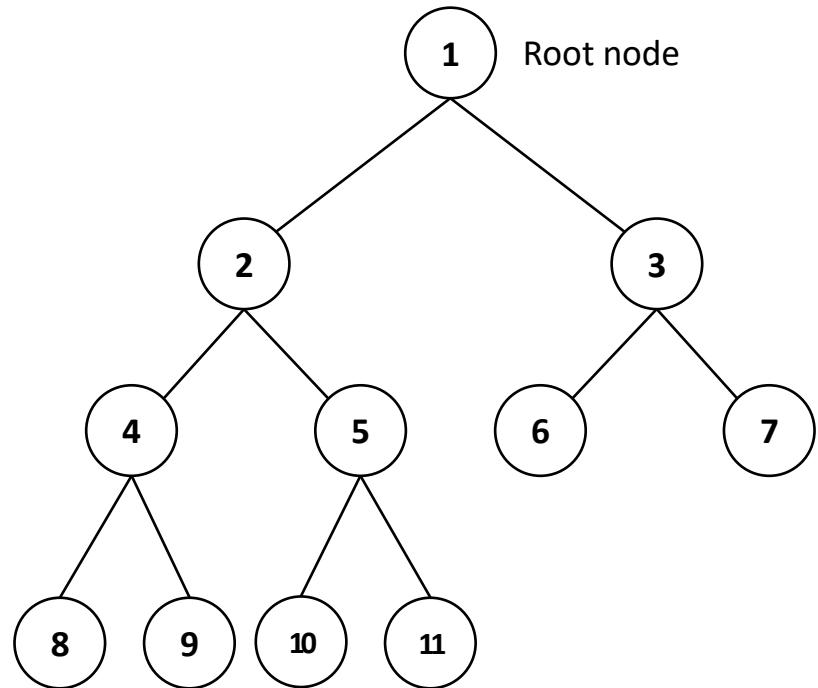


C - data structure

Data Structure

A data structure is a way of collecting and organizing data so that it can be stored and used effectively. When it comes to data structures, there are two different types: linear and non-linear. The former links data in a sequential manner, while the latter links data in a contiguous manner.

- Nonlinear data structure types - tree
 - A data structure that expresses layered relationships with a data structure consisting of nodes and edges connecting them.
 - Root node : top level node
 - Parent node : upper-level node
 - E.g., the parent node of 2 and 3 are 1.
 - Child node : lower level node
 - E.g., the child node of 1 is 2.
 - Leaf node : the lowest level node
 - ex) 8,9,10,11,6,7 are leaf nodes
 - Internal node : all nodes except leaf nodes



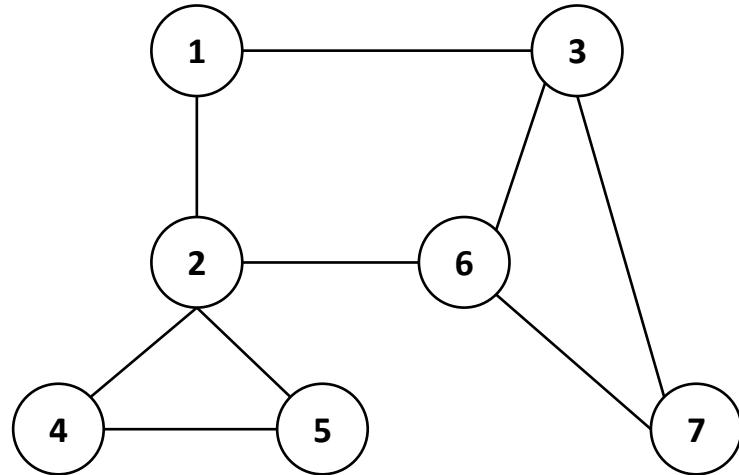
C - data structure

Data Structure

A data structure is a way of collecting and organizing data so that it can be stored and used effectively. When it comes to data structures, there are two different types: linear and non-linear. The former links data in a sequential manner, while the latter links data in a contiguous manner.

- Nonlinear data structure types - graph

- A data structure that expresses relationships between related objects using nodes and edges
- Node : also called vertex
- Edge: a line connecting nodes.
- Use as a network model
- Have no concept of parent-child relationships or the root node
- Graph types : Directed, undirected graphs



C - data structure

Data Structure

- Example 1 of the data structure (queue) - (1/3)
 - Source file - "queue1.c"

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

int queue[SIZE];
int top = 0;

void queue_push();
void queue_pop();
void queue_print();

void main() {
    int input;
    while (1){
        printf("\n\n1.Push (max 5) \n2.pop\n");
        scanf("%d", &input);
        switch (input) {
            case 1:queue_push(); break;
            case 2:queue_pop(); break;
        }
        queue_print();
    }
}
```

C - data structure

Data Structure

- Example 1 of the data structure (queue) - (2/3)
 - Source file - "queue1.c"

```
void queue_push() {
    int n, i;
    if (top < SIZE){
        top++;
        for (i = top - 1; i > 0; i--) {
            queue[i] = queue[i - 1];
        }
        printf("\ninput : ");
        scanf("%d", &n);
        queue[0] = n;
    }
    else{
        printf("Queue overflow\n");
    }
}
```

C - data structure

Data Structure

- Example 1 of the data structure (queue) - (3/3)
 - Source file - "queue1.c"

```
void queue_pop() {
    if (top == 0) {
        printf("Empty\n\n");
    }
    else
    {
        top--;
        printf("pop : %d\n\n", queue[top]);
    }
}

void queue_print() {
    int i;
    for (i = 0; i < top; i++)
        printf("%d ", queue[i]);
}
```

- Example 2 of the data structure (stack) - (1/3)

- Source file - "stack1.c"

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 5

int stack[SIZE];
int top = 0;

void stack_push();
void stack_pop();
void stack_print();

void main() {
    int input;
    while (1){
        printf("\n\n1.Push (max 5) \n2.pop\n");
        scanf("%d", &input);
        switch (input) {
        case 1:stack_push(); break;
        case 2:stack_pop(); break;
        }
        stack_print();
    }
}
```

C - data structure

Data Structure

- Example 2 of the data structure (stack) - (2/3)
 - Source file - "stack1.c"

```
void stack_push() {
    int n;
    if (top >= SIZE){
        printf("Stack overflow\n\n");
    }
    else
    {
        printf("\ninput : ");
        scanf("%d", &n);
        stack[top++] = n;
    }
}
```

C - data structure

Data Structure

- Example 2 of the data structure (stack) - (3/3)
 - Source file - "stack1.c"

```
void stack_pop() {
    if (top == 0) {
        printf("Empty\n\n");
    }
    else
    {
        top--;
        printf("pop : %d\n\n", stack[top]);
    }
}

void stack_print() {
    int i;
    for (i = 0; i < top; i++)
        printf("%d ", stack[i]);
}
```

C - data structure

Data Structure

- Assignment : data structure
 - Implement the linear data structure below using a linked list.
 - Linear data structures
 - Queue - FIFO implementation
 - Stack - LIFO implementation
 - Must use insert and delete functions required for each data structure
 - Utilize structs and pointers
 - Implement the above with an eye on overflows

03

Python programming

- Python programming overview
- Python programming environment setup
- Structure of the Python program
- Python - variables and data types
- Python - operator
- Python - data input/output
- Python - control statements
- Python - functions
- Python - class
- Python - modules and packages
- Python - exception
- Python - debugging

Python programming overview

Python overview

Python is a high-level programming language released in 1991 by programmer Guido van Rossum. The name Python is said to come from Guido's favorite comedy series, Monty Python's Flying Circus.

- Python
 - Dictionary meaning : a giant serpent that appeared in Greek mythology
 - Origin of the logo, which is a reminiscent of a pair of snakes
 - Snakes are often depicted in related textbooks and books.
 - Key features
 - Platform independent
 - Interpreter language
 - Object-oriented language
 - Support for dynamic typing



python™

Python programming overview

Python overview

We will learn the key features of Python.

- Python
 - Key features (organized)

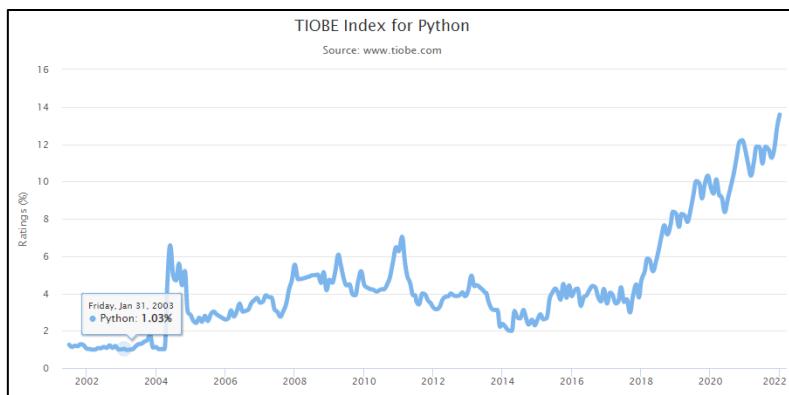
Technological trend	Description
Platform independent	<ul style="list-style-type: none">• Does not depend on a specific operating system or machine and can be used independently• Opposite: platform dependent
Interpreter language	<ul style="list-style-type: none">• A computer program or environment that directly executes the source code of a programming language• Read and execute code commands one line at a time
Object-oriented language	<ul style="list-style-type: none">• A language that attempts to move away from thinking of a computer program as a list of commands and instead thinks of it as a collection of independent units, or "objects"
Support for dynamic typing	<ul style="list-style-type: none">• When specifying a variable, a user doesn't have to specify its data type, etc., and the computer interprets it by itself.

Python programming overview

Python overview

TIOBE, a software code quality management company, publishes a monthly ranking of programming language popularity, and as of November 2023, Python is ranked #1 overall.

- Python
 - Programming language rankings (as of November 2023)
 - Current rank : #1



Nov 2023	Nov 2022	Change	Programming Language	Ratings	Change
1	1		Python	14.16%	-3.02%
2	2		C	11.77%	-3.31%
3	4	▲	C++	10.36%	-0.39%
4	3	▼	Java	8.35%	-3.63%
5	5		C#	7.65%	+3.40%
6	7	▲	JavaScript	3.21%	+0.47%
7	10	▲	PHP	2.30%	+0.61%
8	6	▼	Visual Basic	2.10%	-2.01%
9	9		SQL	1.88%	+0.07%
10	8	▼	Assembly language	1.35%	-0.83%

Source: <https://www.tiobe.com/tiobe-index/>

Python programming overview

Python overview

One of Python's greatest strengths is its seamless integration with a wide variety of programming languages. This has led to the creation of many different arrays of Python.

- Types of Python
 - Cpython
 - Python written in C
 - By default, when we say Python, we mean Cpython.
 - Jython
 - Python implemented in Java
 - It runs on the Java Virtual Machine (JVM).



Sources: <http://egloos.zum.com/incredible/v/7410010>, <https://www.rexconsulting.net/jython-jmx-monitoring.html>

Python programming overview

Python overview

One of Python's greatest strengths is its seamless integration with a wide variety of programming languages. This has led to the creation of many different arrays of Python.

- Types of Python
 - IronPython
 - Python implemented in C#
 - Developed for .NET and Mono
 - PyPy
 - Python implemented by itself
 - The aim is to outperform CPython in terms of speed.



Sources: <https://ko.wikipedia.org/wiki/IronPython>, <https://ko.wikipedia.org/wiki/PyPy>

Python programming overview

Python overview

There are many things you can do with Python. It does what most programming languages do easily and cleanly. We will look at some of the most common examples.

- What you can do with Python
 - Creating system utilities
 - You have the tools to use the system commands of your operating system (Windows, Linux, etc.), so you have an advantage in creating system utilities based on them.
 - ※ Utilities : software that helps you use your computer
 - GUI programming
 - Python is well equipped with tools for GUI programming
 - ※ Good example : Tkinter
 - Combining with other languages
 - For example, programs written in C or C++ can be used in Python, and programs written in Python can be used in C or C++.
 - Other things
 - Web programming, numerical computing programming, database programming, etc.

Python programming overview

Python overview

Python 3.x should be used with caution as it does not support backward compatibility with Python 2.x.

- Changes over time in Python
 - Differences between 2.x and 3.x

Division	2.x	3.x
Change print to a function	>>> print "Hello World" Hello World	>>> print ("Hello World") Hello World >>> print "Hello World" SyntaxError: invalid syntax
Long datatype is lifted, replaced by int	>>>type(2**31) <type 'long'> >>>sys.maxint 2147483647 (maxint and above will be of a long type)	>>>type(2**31) <class 'int'> >>>type(2**40) <class 'int'> (all maxint and above are treated as int)
Treat the result of an int/int as a float	>>>3 / 2 1	>>>3 / 2 1.5 >>>3 // 2 1
Change String, Unicode Scheme	A plain string is a string with an encoding and Unicode exists separately	A plain string is equivalent to Unicode and a string with an encoding is expressed in bytes

Python programming environment setup

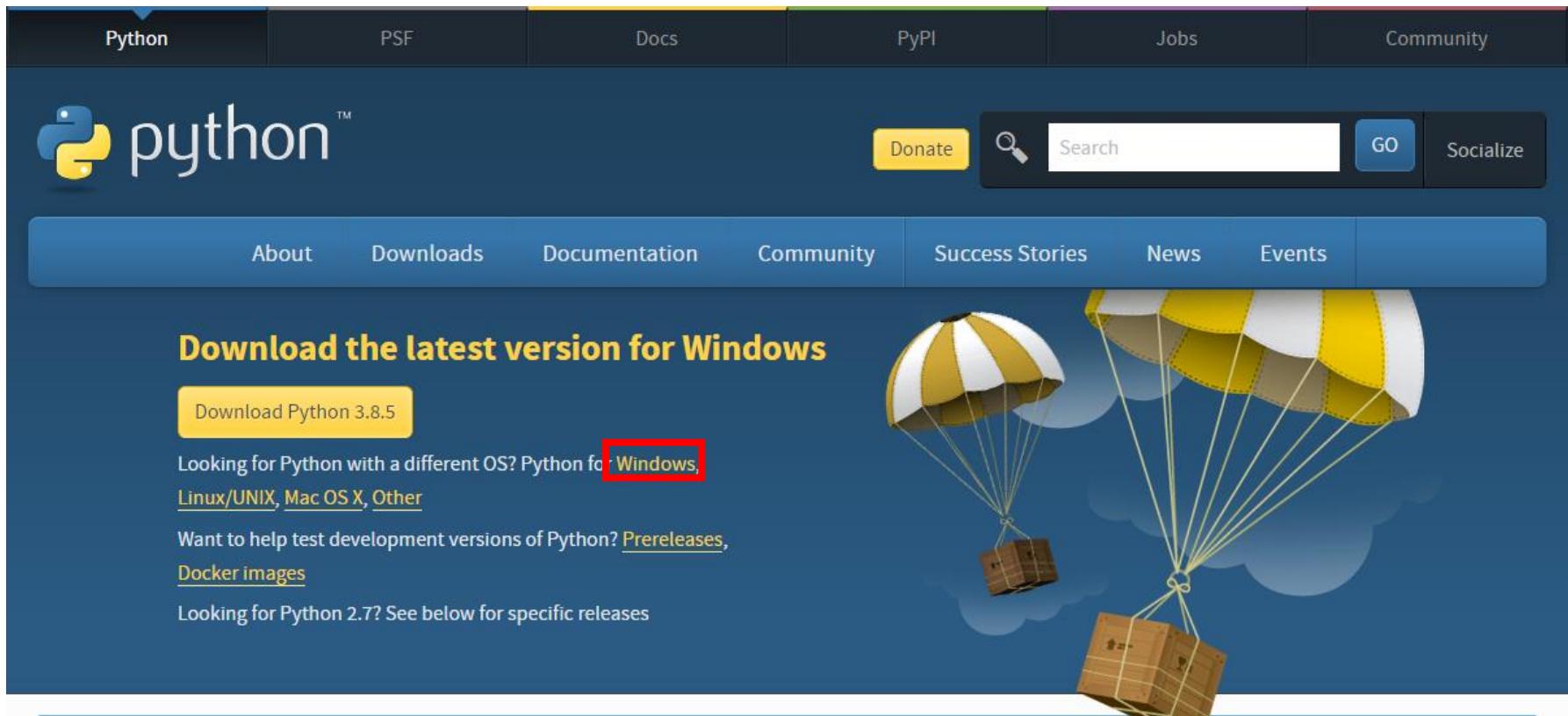
How to install Python

- Lab environments
 - Operating system: Windows 10
 - Programs used: Python 3.8.5 (Latest), Pycharm 2023.3.2 (Latest)

Python programming environment setup

How to install Python

- How to install Python3
 - Visit the download page at <https://www.python.org/downloads/> and click “Windows” below



Python programming environment setup

How to install Python

- How to install Python3
 - Select and download the 64-bit exe install file.

Python Releases for Windows

- [Latest Python 3 Release - Python 3.8.5](#)
- [Latest Python 2 Release - Python 2.7.18](#)

Stable Releases

- [Python 3.8.5 - July 20, 2020](#)

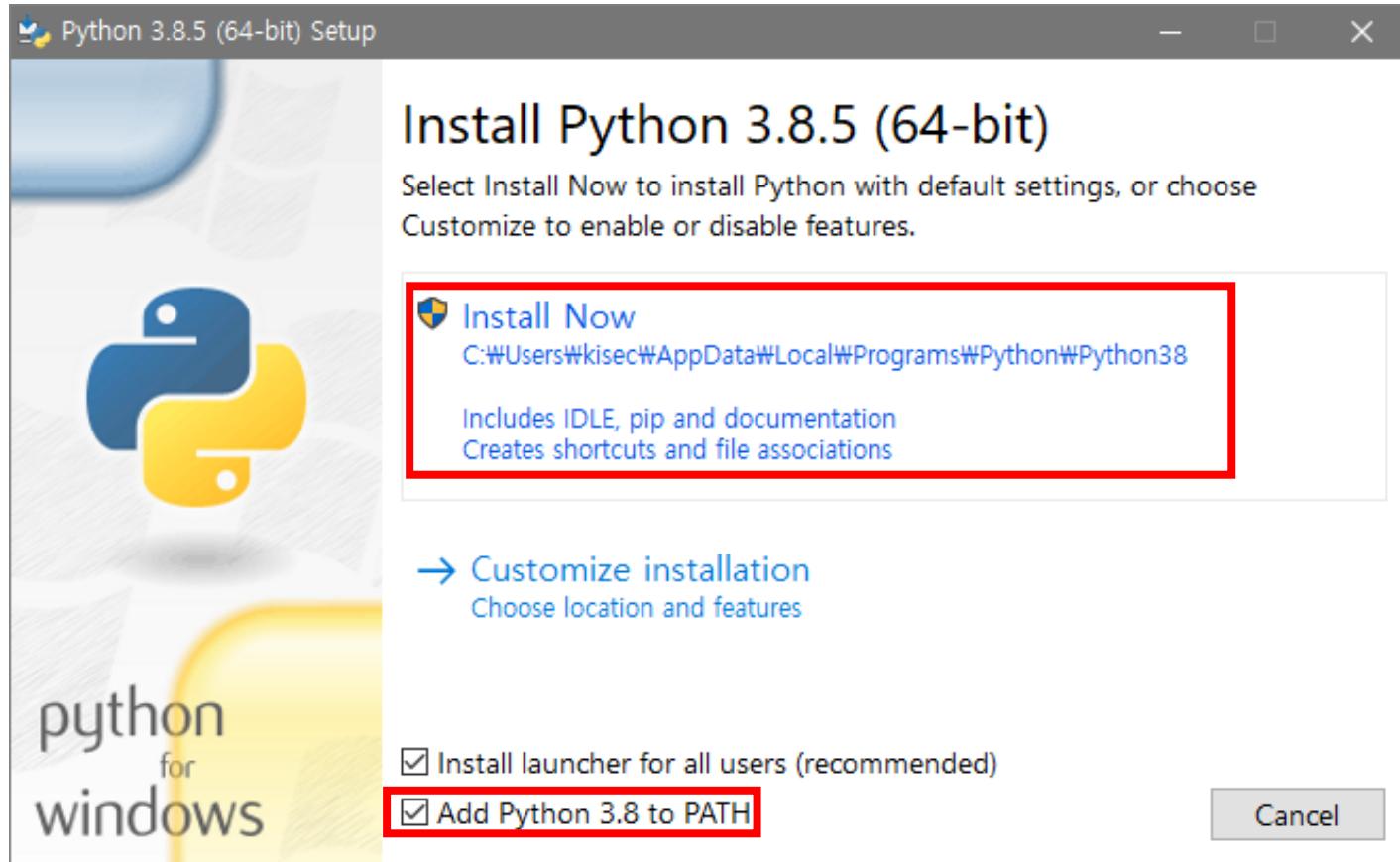
Note that Python 3.8.5 *cannot* be used on Windows XP or earlier.

- Download [Windows help file](#)
- Download [Windows x86-64 embeddable zip file](#)
- Download [Windows x86-64 executable installer](#)
- Download [Windows x86-64 web-based installer](#)
- Download [Windows x86 embeddable zip file](#)
- Download [Windows x86 executable installer](#)
- Download [Windows x86 web-based installer](#)

Python programming environment setup

How to install Python

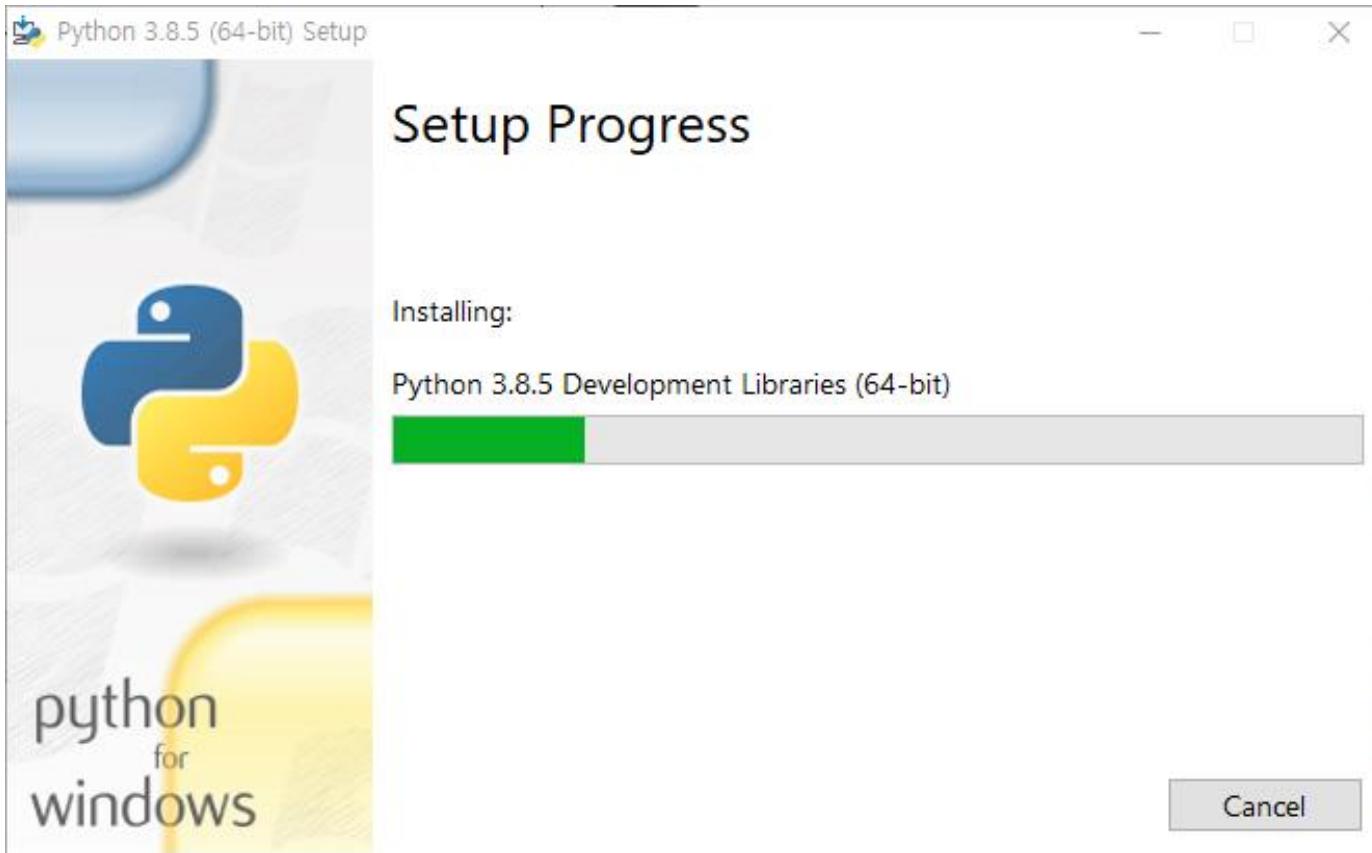
- How to install Python3
 - Run `python-x.x.x.x.exe` -> Check the “Add Python x.x to PATH” box -> Click “Install Now”.



Python programming environment setup

How to install Python

- How to install Python3
 - Click “Yes” to “Do you want to allow this app to make changes to your device?” question. -> Proceed with installation.



Python programming environment setup

How to install Python

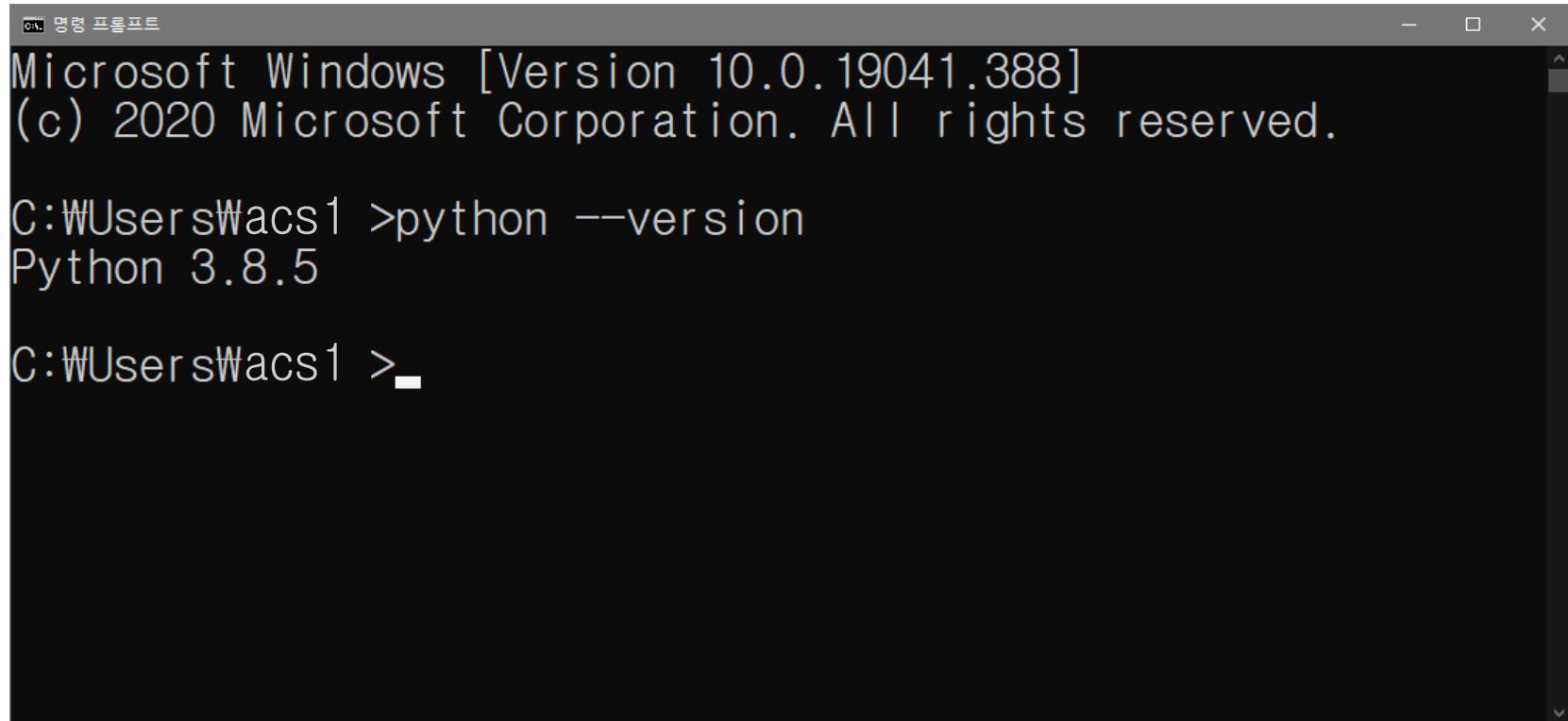
- How to install Python3
 - In the dialog box that informs you that the installation is complete, click the "Close" button.



Python programming environment setup

How to install Python

- How to install Python3
 - Check if Python is installed -> press Ctrl + r -> type "cmd" -> type "python --version" -> check the version.
 - If you did not check the "Add Python 3.8 to PATH" box during installation, you will get an error in this process.



A screenshot of a Windows Command Prompt window titled "명령 프롬프트". The window shows the following text:
Microsoft Windows [Version 10.0.19041.388]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\Wacs1>python --version
Python 3.8.5

C:\Users\Wacs1>

Python programming environment setup

Python IDE overview

PyCharm is an IDE for Python developed by JetBrains r.s.o. and is one of the most mature Python development tools available. There is a Community Edition (free) and a Professional Edition (paid) for PyCharm.

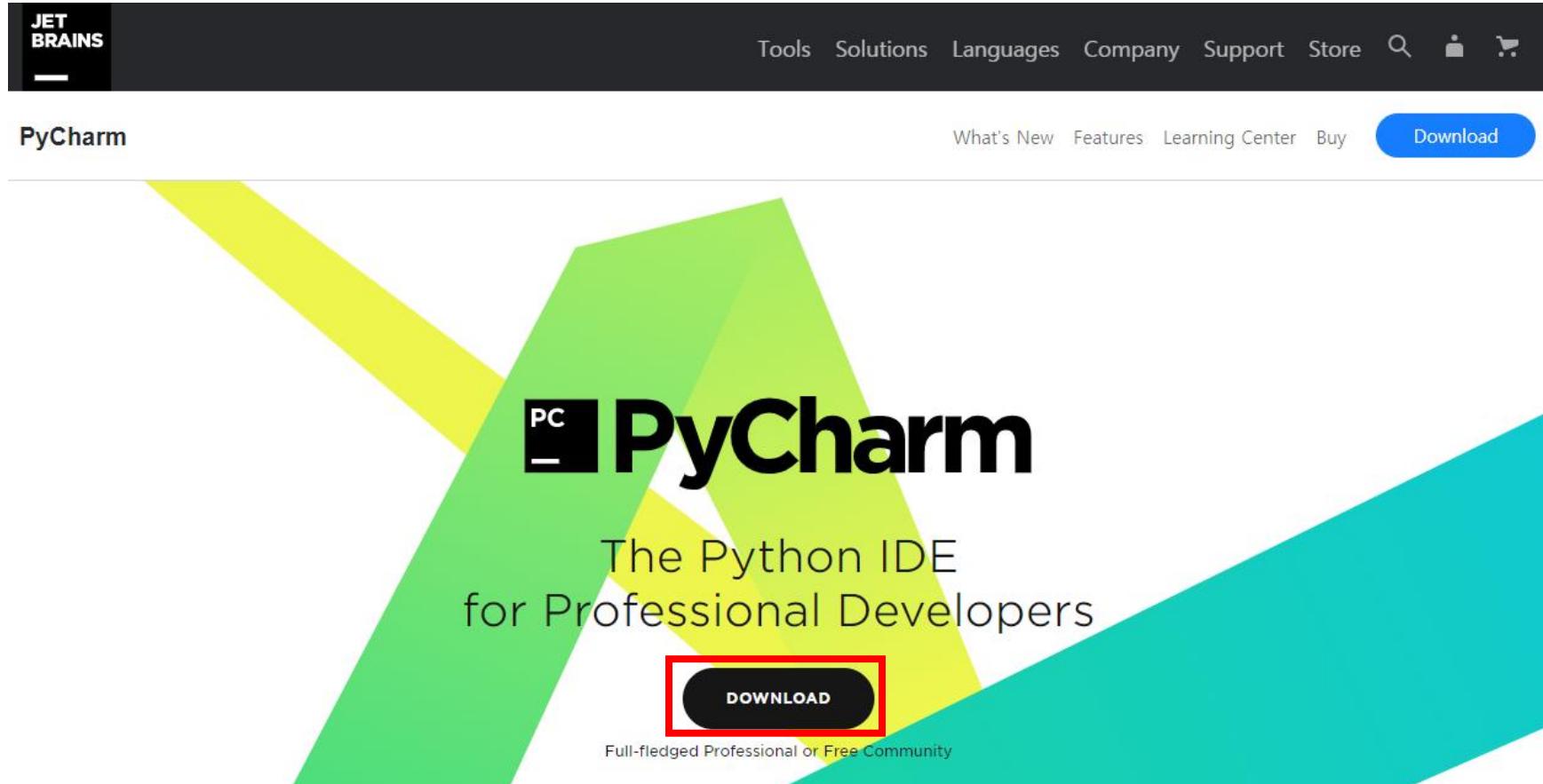
- PyCharm
 - IDE for Python developed by JetBrains r.s.o.
 - Provide code auto-completion
 - Native support for package isolation environments such as VirtualEnv and Anaconda
 - Provide an API for developers to extend PyCharm functionality by creating their own plugins
 - Support integrated Python debugger, project, and code exploration



Python programming environment setup

How to install the Python IDE

- How to install PyCharm
 - Go to the download page at <https://www.jetbrains.com/pycharm/> and click the "DOWNLOAD" button.



Python programming environment setup

How to install the Python IDE

- How to install PyCharm
 - Check the Windows -> Click the “Download” button for the Community version.

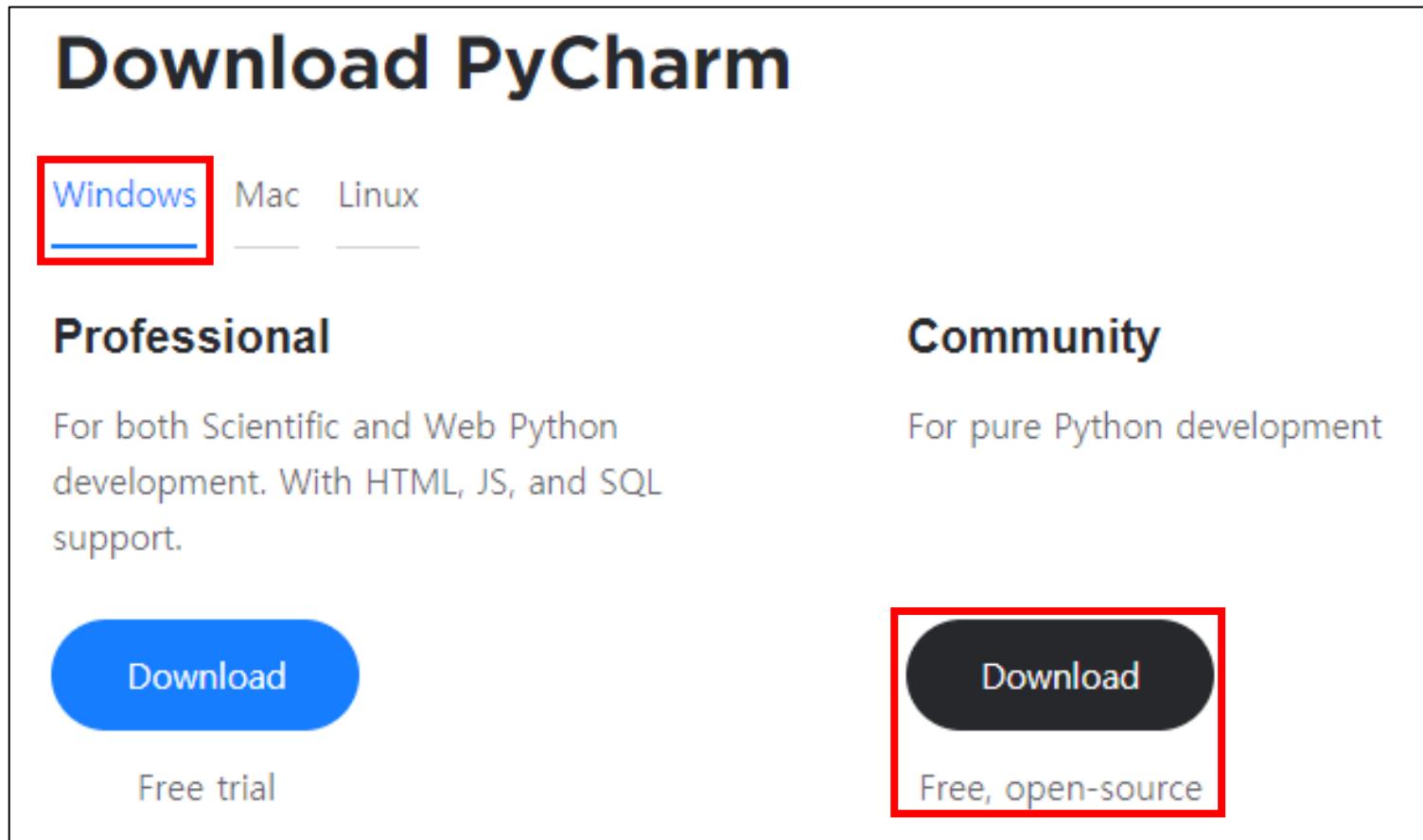
Download PyCharm

Windows Mac Linux

Professional
For both Scientific and Web Python development. With HTML, JS, and SQL support.

Community
For pure Python development

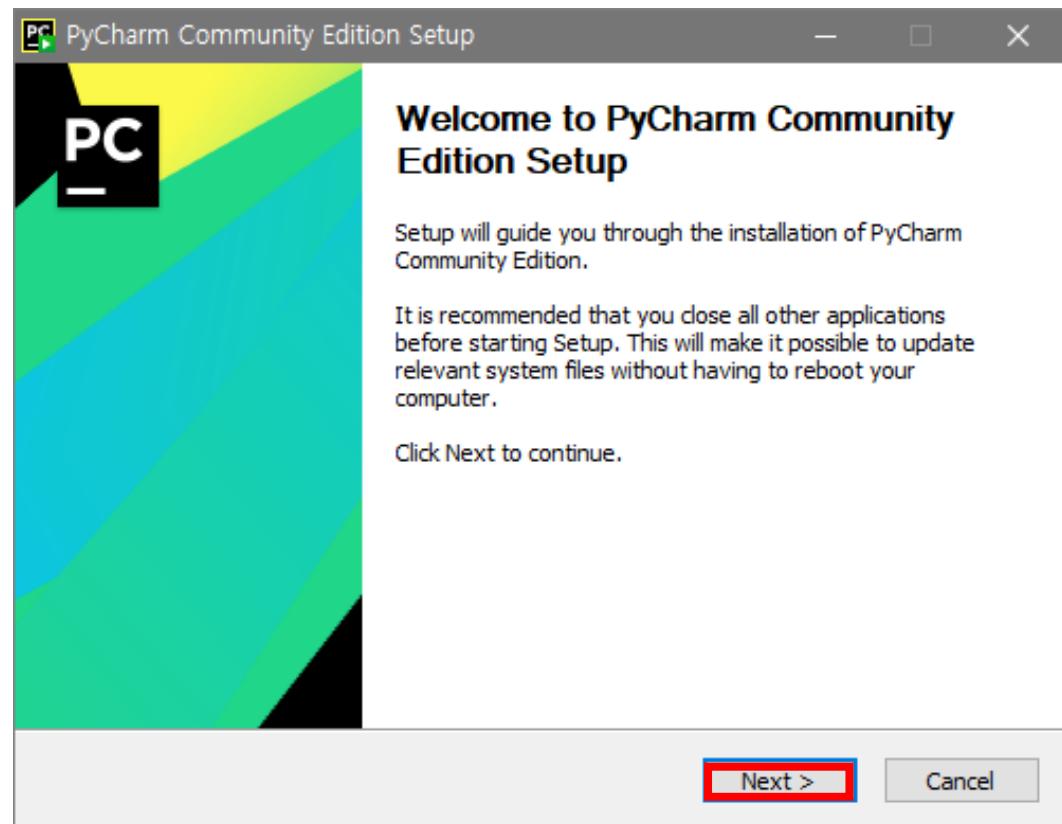
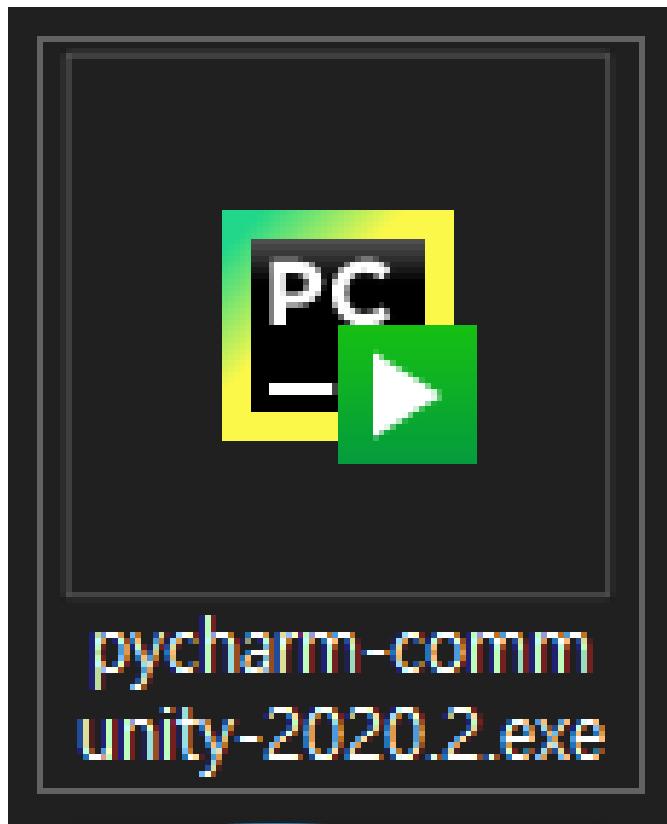
[Download](#) [Free trial](#) [Download](#) [Free, open-source](#)



Python programming environment setup

How to install the Python IDE

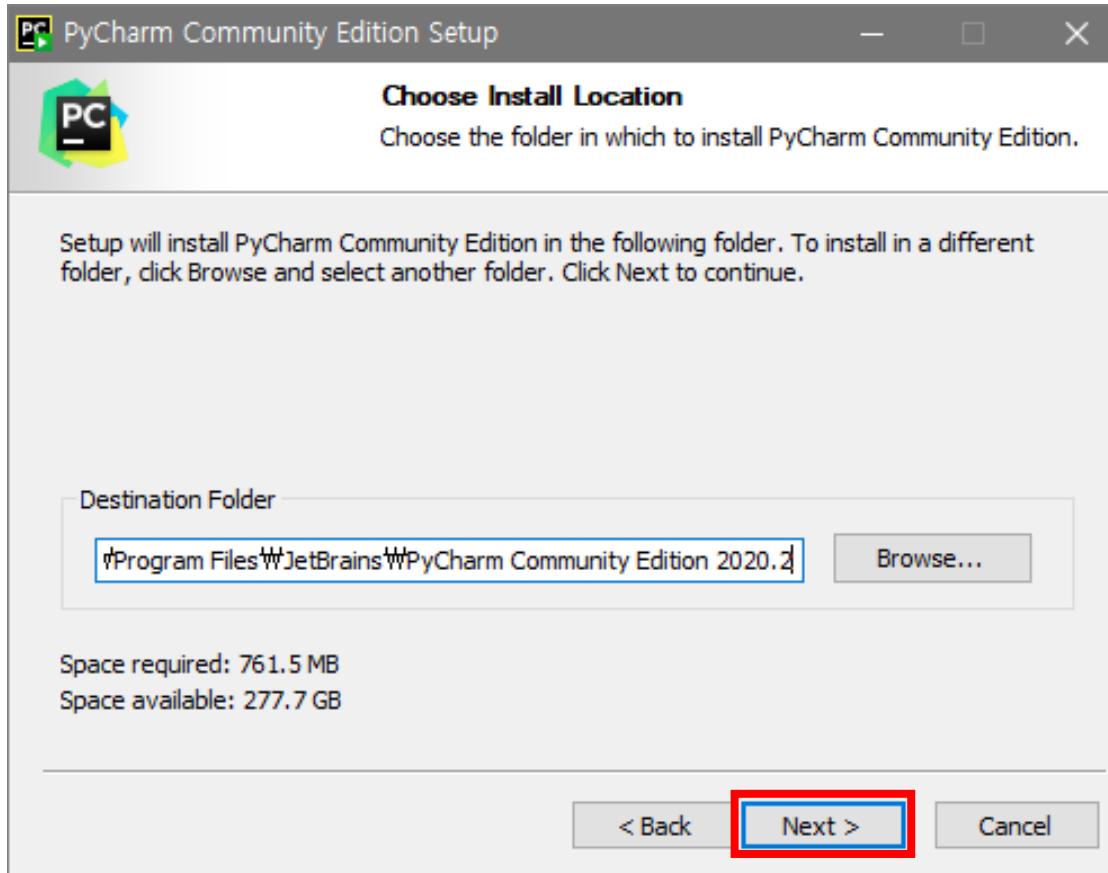
- How to install PyCharm
 - Check the downloaded .exe file -> Run it -> Click "Yes" to "Do you want to allow this app to make changes to your device?" question. -> Click the "Next>" button on the Setup dialog box.



Python programming environment setup

How to install the Python IDE

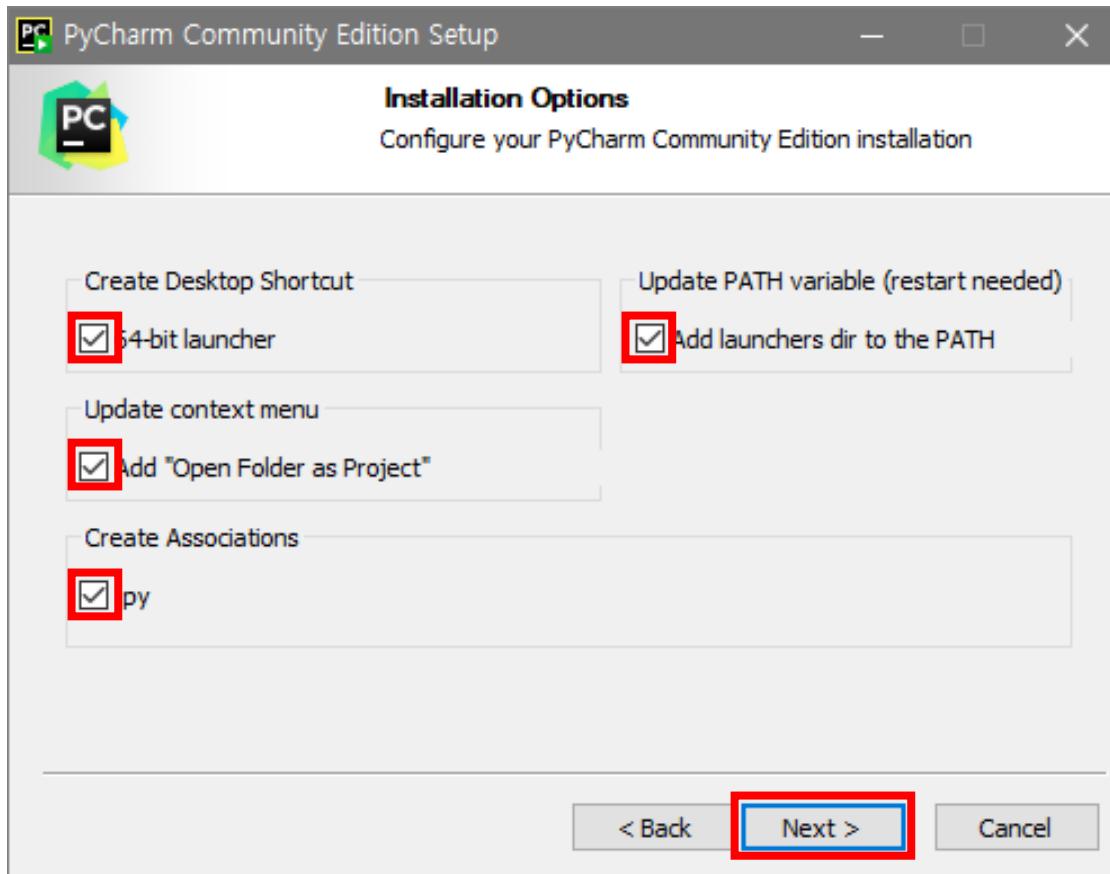
- How to install PyCharm
 - Keep the default installation path -> click the “Next>” button.



Python programming environment setup

How to install the Python IDE

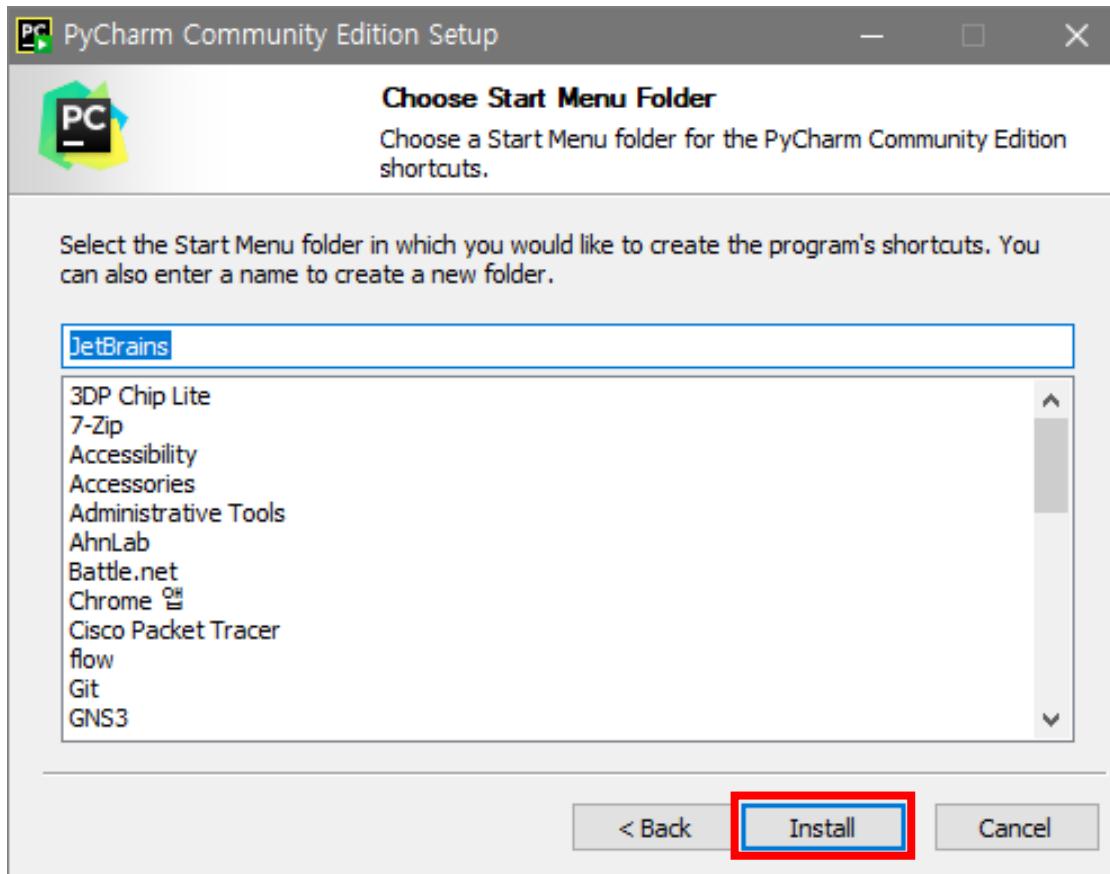
- How to install PyCharm
 - Check all Installation Options boxes -> click the “Next>” button.



Python programming environment setup

How to install the Python IDE

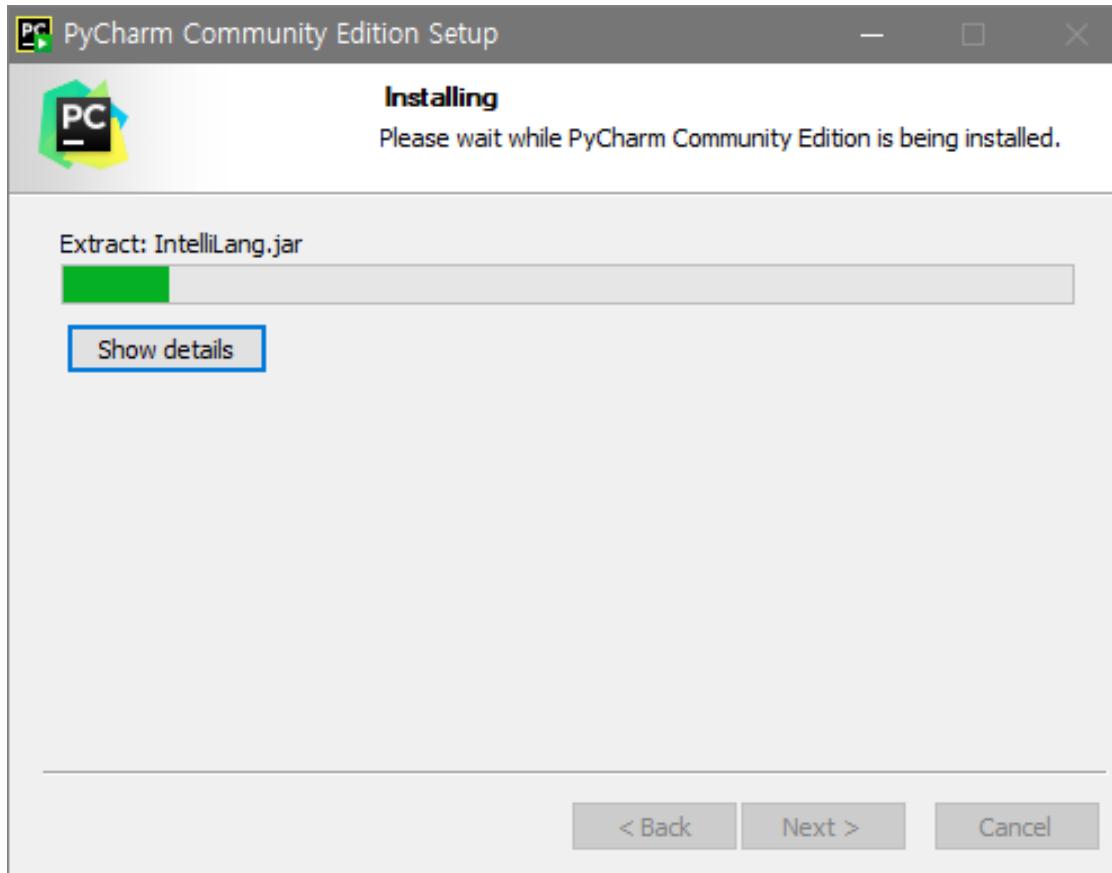
- How to install PyCharm
 - Click the “Install” box.



Python programming environment setup

How to install the Python IDE

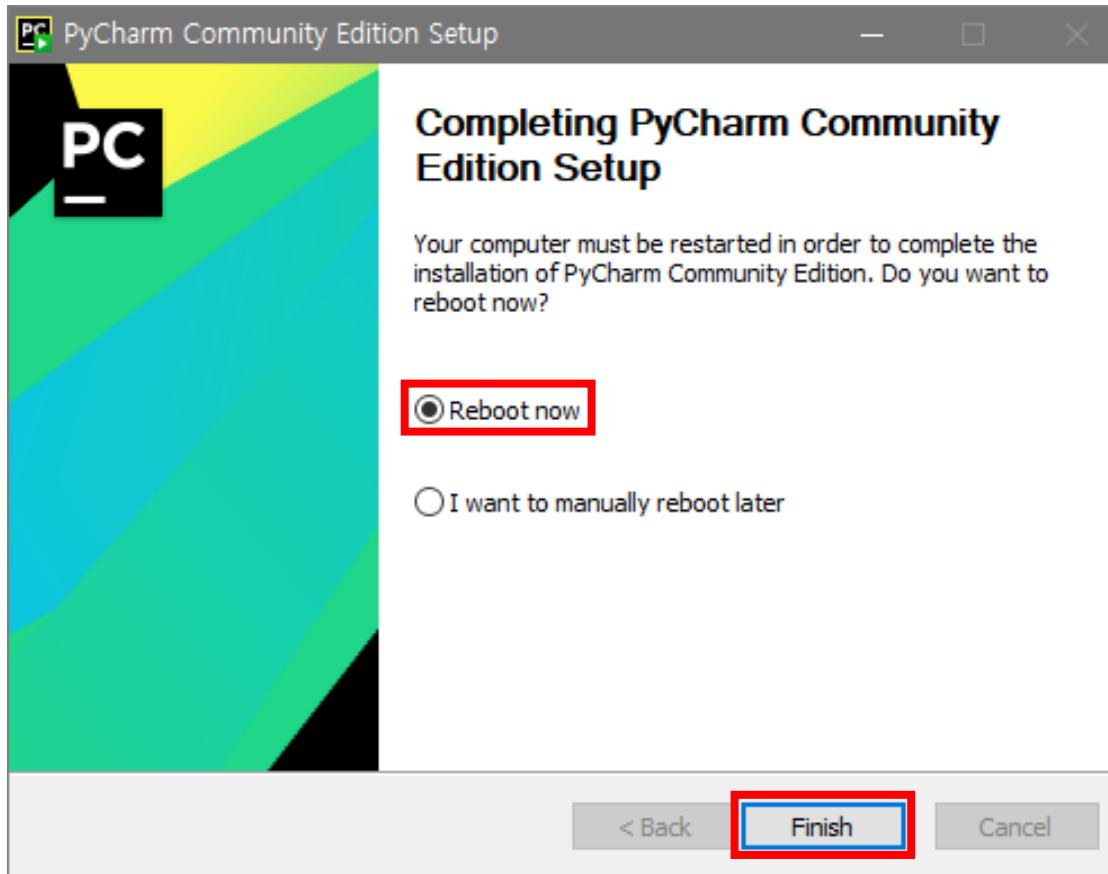
- How to install PyCharm
 - Proceed with the installation.



Python programming environment setup

How to install the Python IDE

- How to install PyCharm
 - Check “Reboot now” -> click the “Finish” button -> restart your computer.

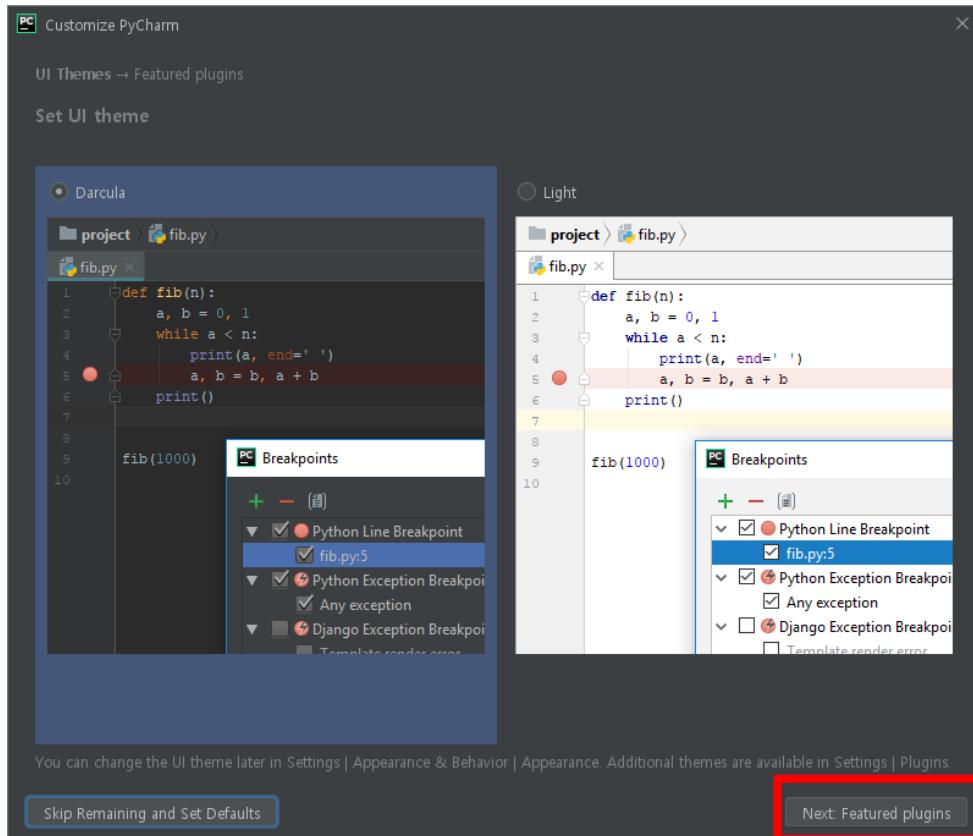


Python programming environment setup

How to install the Python IDE

- How to install PyCharm

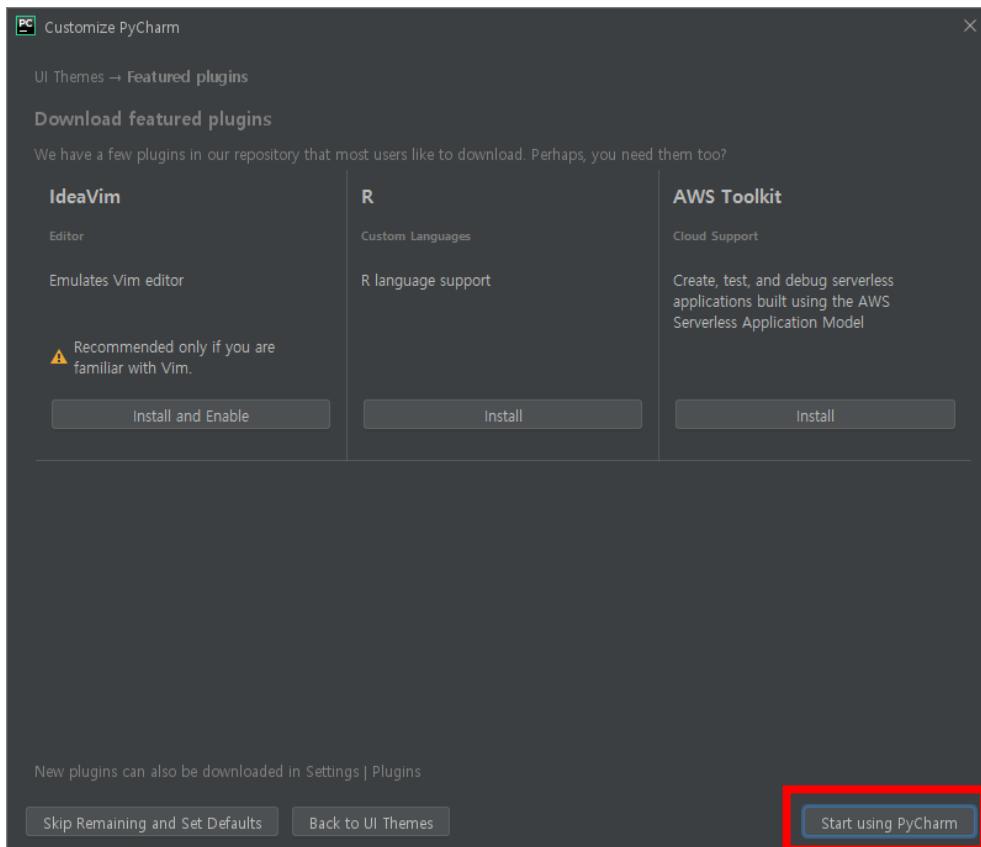
- Once PyCharm is installed, click its icon to run it -> Select the UI theme -> click the “Next Featured plugins” button.



Python programming environment setup

How to install the Python IDE

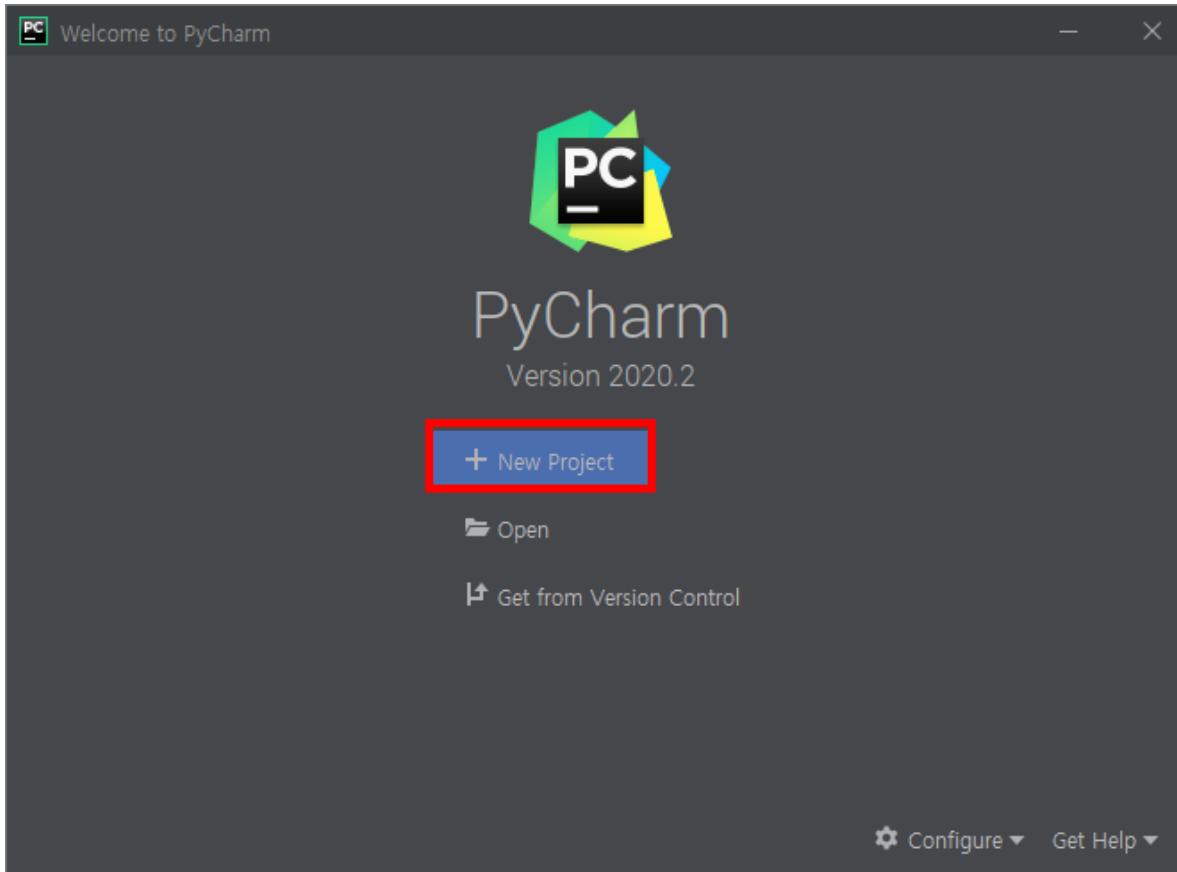
- How to install PyCharm
 - Select the Featured plugin(s) if needed and click the “Install” button(s) -> click the “Start using PyCharm” button.



Python programming environment setup

How to install the Python IDE

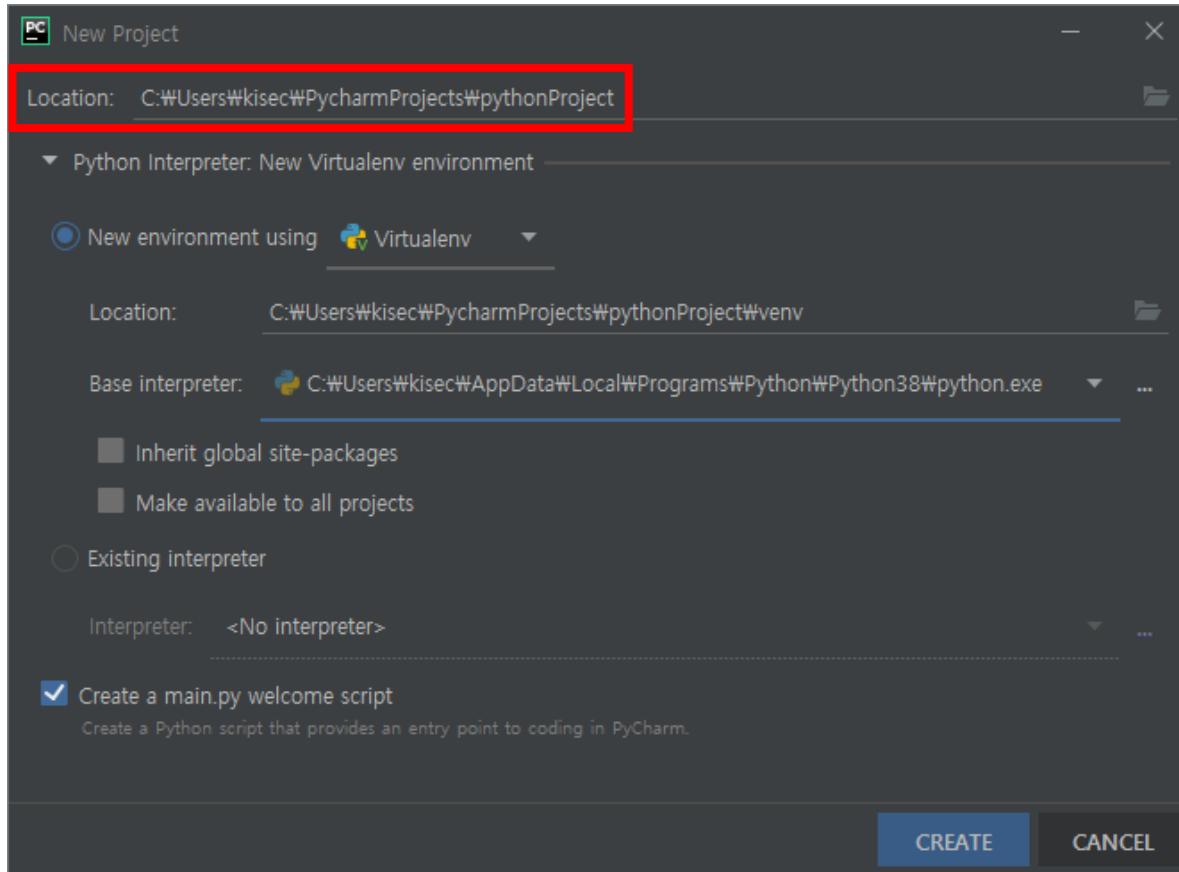
- Python3 & PyCharm Integration
 - In the Welcome to PyCharm dialog box, click the “New Project” button.



Python programming environment setup

How to install the Python IDE

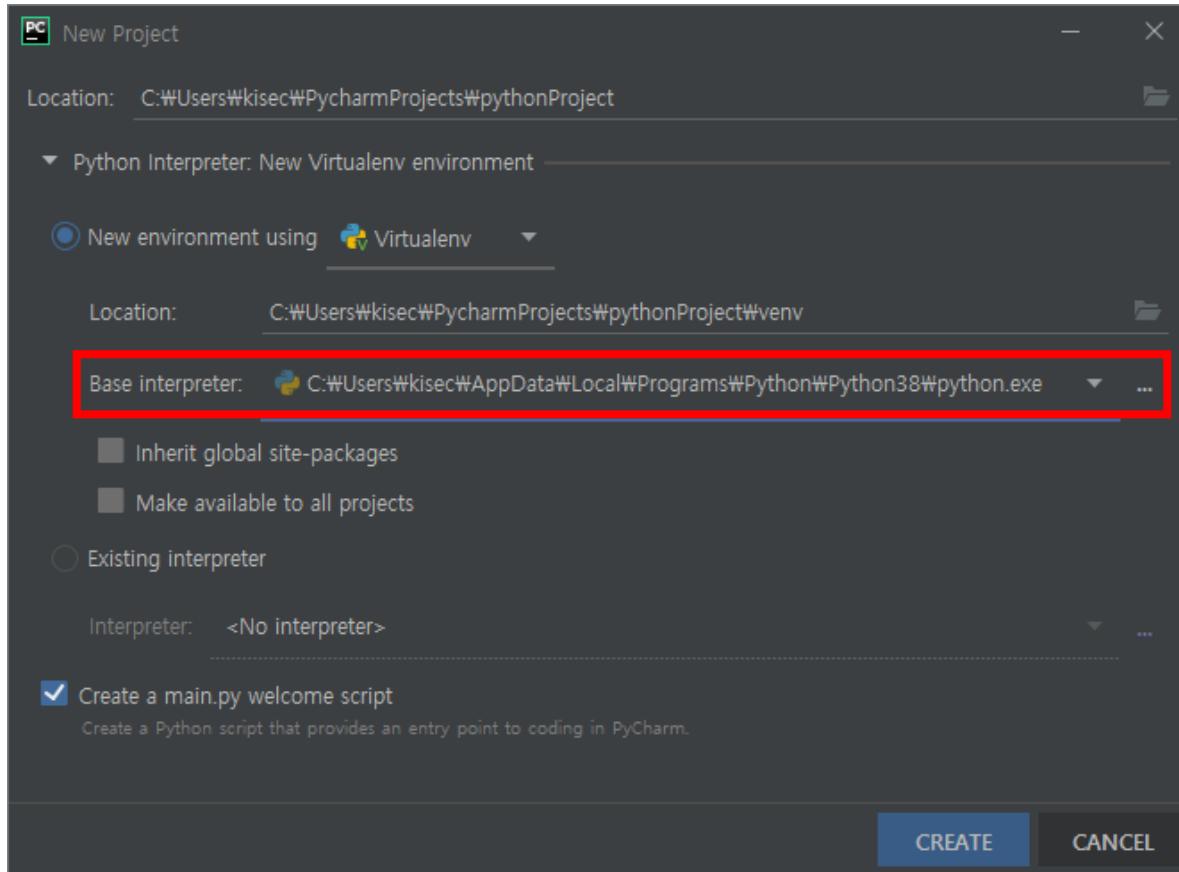
- Python3 & PyCharm integration
 - Location: Set the desired location and the project name.



Python programming environment setup

How to install the Python IDE

- Python3 & PyCharm integration
 - Set the Base Interpreter.



Python programming environment setup

How to install the Python IDE

- Python3 & PyCharm integration
 - Set the Base Interpreter
 - Press the Windows key -> type cmd -> run "where python" command -> copy the first path.

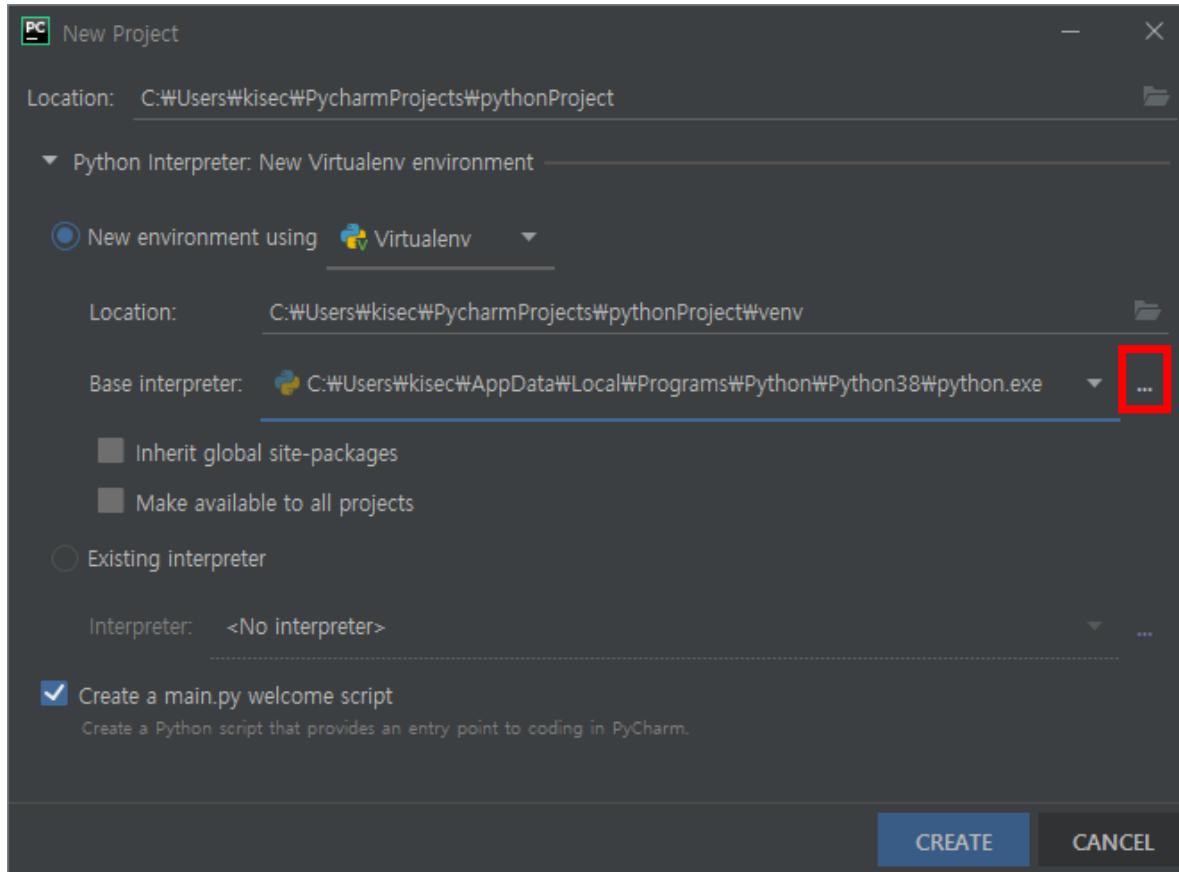
```
C:\Users\wsuyoung>where python
C:\Users\wsuyoung\(userData)\Local\Programs\Python\Python38\python.exe
C:\Users\wsuyoung\anaconda3\python.exe
C:\Users\wsuyoung\userData\Local\Microsoft\WindowsApps\python.exe

C:\Users\wsuyoung>
```

Python programming environment setup

How to install the Python IDE

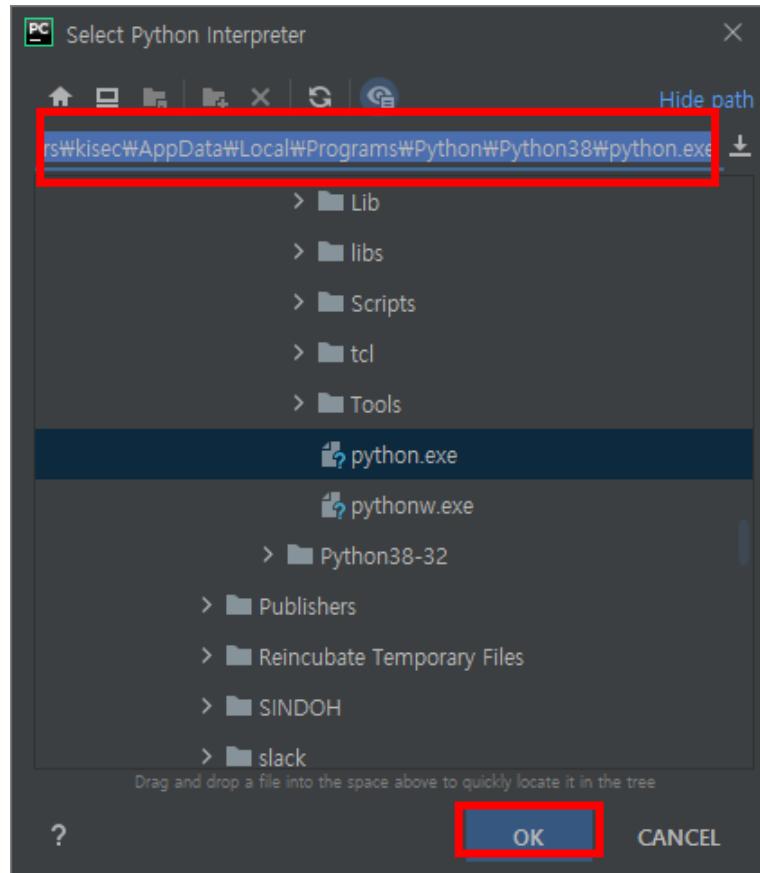
- Python3 & PyCharm integration
 - At the right end of the Base Interpreter address bar, click the "... (Browse)" icon.



Python programming environment setup

How to install the Python IDE

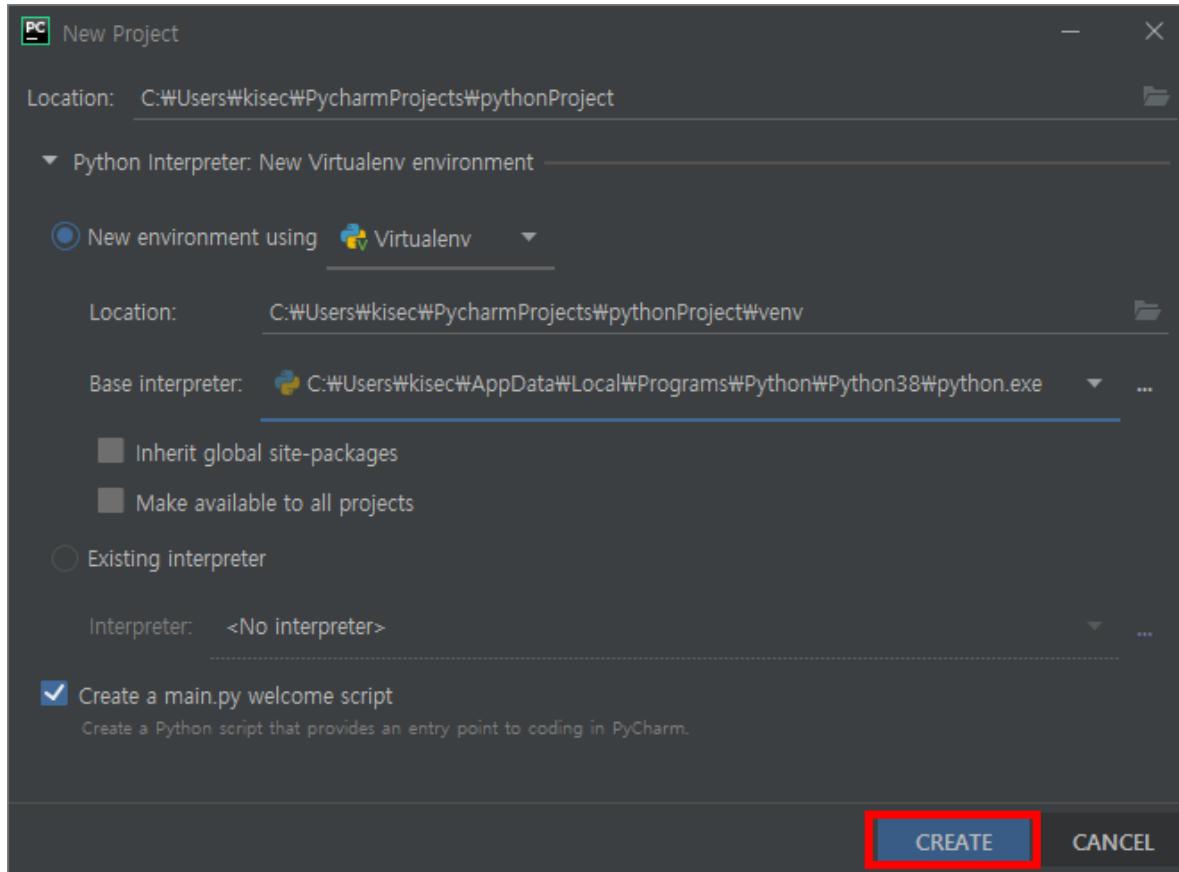
- Python3 & PyCharm integration
 - Paste the copied destination path -> click the “OK” button.



Python programming environment setup

How to install the Python IDE

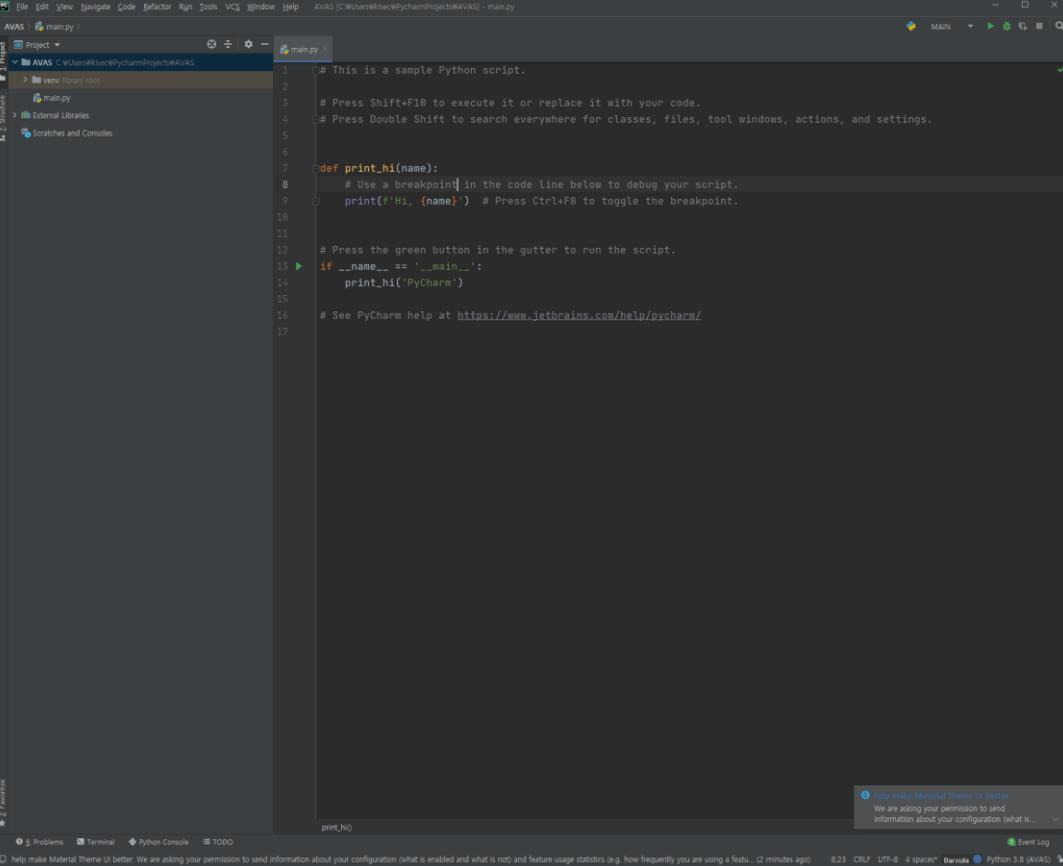
- Python3 & PyCharm integration
 - After all settings are complete, click the “CREATE” button.



Python programming environment setup

How to install the Python IDE

- Python3 & PyCharm integration
 - Check to see if the project pane is working properly.



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The title bar indicates the project is 'AVAS' and the file is 'main.py'. The left sidebar displays the 'Project' view with a tree structure showing 'AVAS' and 'main.py'. The main code editor pane contains the following Python code:

```
# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.

def print_hi(name):
    # Use a breakpoint! in the code line below to debug your script.
    print(f'Hi, {name}!') # Press Ctrl+F8 to toggle the breakpoint.

# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    print_hi('PyCharm')

# See PyCharm help at https://www.jetbrains.com/help/pycharm/
```

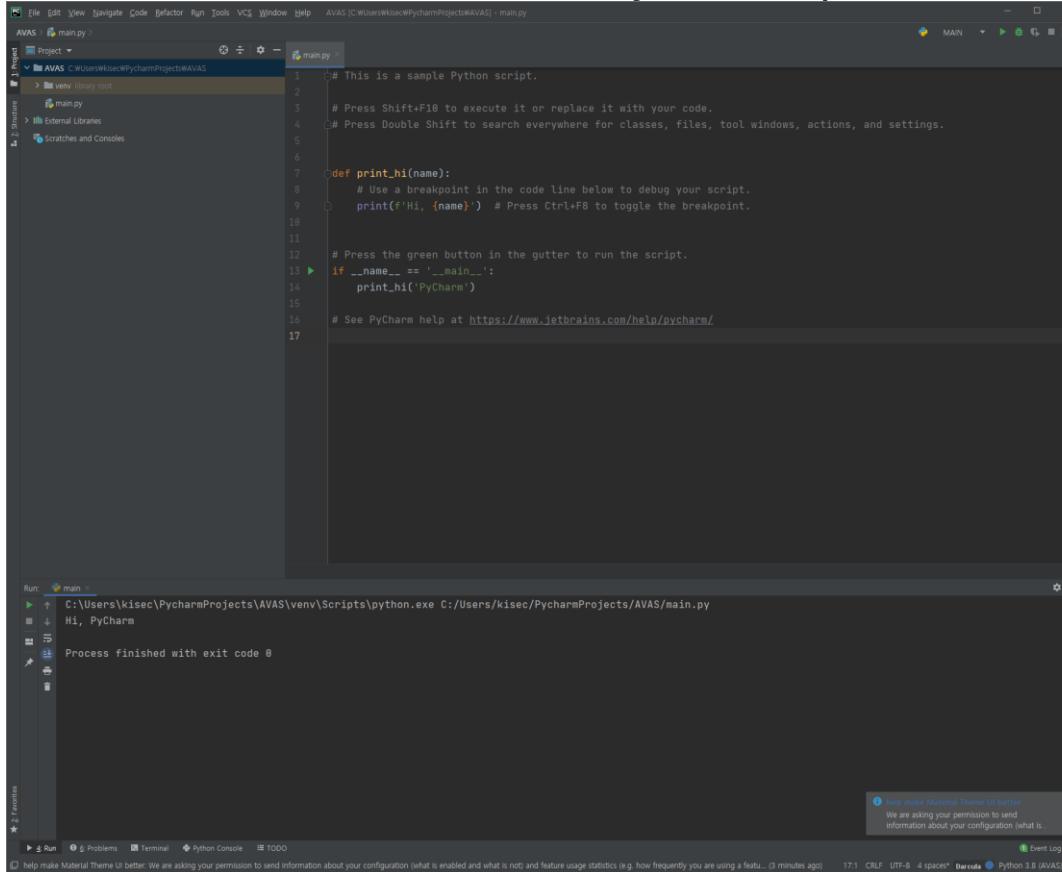
The status bar at the bottom shows 'print_hi()' and 'Event Log'. A small pop-up window in the bottom right corner asks for permission to send information about the configuration.

Python programming environment setup

How to install the Python IDE

- Python3 & PyCharm integration

- Press Ctrl + Shift + F10 to run or choose and click Run ‘the project name’ in the top menu bar
-> check the execution status at the bottom (Hi, PyCharm is printed).



```
# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.

def print_hi(name):
    # Use a breakpoint in the code line below to debug your script.
    print(f'Hi, {name}!') # Press Ctrl+F8 to toggle the breakpoint.

# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    print_hi('PyCharm')

# See PyCharm help at https://www.jetbrains.com/help/pycharm/
```

The screenshot shows the PyCharm interface with a dark theme. The main window displays a Python file named 'main.py'. The code in the file prints 'Hi, PyCharm'. Below the code editor, the 'Run' tool window shows the output of running the script: 'Hi, PyCharm' and 'Process finished with exit code 0'. At the bottom of the screen, there is a status bar with various icons and text, including a message about helping make Material Theme UI better.

Structure of the python program

Keywords and identifiers

Programs use identifiers to distinguish variables, functions, and so on, and identifiers cannot be named after keywords, which are special features of Python.

- What is a keyword?
 - Strings already reserved in Python
 - Cannot be used for any other purpose

Python 3.10 keyword list

False	None	True	__peg_parser__
and	as	assert	async
await	break	class	continue
def	del	elif	else
except	finally	for	from
global	if	import	in
is	lambda	nonlocal	not
or	pass	raise	return
try	while	with	yield

Structure of the python program

Keywords and identifiers

Programs use identifiers to distinguish variables, functions, and so on, and identifiers cannot be named after keywords, which are special features of Python.

- What is an identifier?
 - A term used only in the specification of a programming language
 - Used to distinguish variables, constants, functions, user-defined types, etc. from others
 - A generalized "name" that includes the names of variables, constants, functions, user-defined types, etc.
 - Can be specified arbitrarily by the user
 - Self-explanatory naming is encouraged.

Structure of the python program

Keywords and identifiers

Programs use identifiers to distinguish variables, functions, and so on, and identifiers cannot be named after keywords, which are special features of Python.

- Different notations for identifiers

Phrase writing format	Description	Example
Camel case	<ul style="list-style-type: none">Capitalize the first letter of each word, but lowercase the letters that precede it	camelVariable, firstName, lastName ...
Pascal case	<ul style="list-style-type: none">Capitalize all the first letters of a word	PascalVariable, FirstName, LastName ...
Hungarian Case	<ul style="list-style-type: none">Datatype prefix notationRarely used these days because there are as many different formats and document data as there are languages	strName, bBusy, szName ...
Snake Case	<ul style="list-style-type: none">Underscores between wordsIt is customary to avoid using identifiers that begin with "_" or "__" because they are often reserved for extensions in many languages.	first_Name, Last_Name, str_name

Source : wikidocs.net

Structure of the python program

Keywords and identifiers

Programs use identifiers to distinguish variables, functions, and so on, and identifiers cannot be named after keywords, which are special features of Python.

- Rules for writing identifiers

- Identifiers can be a combination of lowercase (a to z) or uppercase (A to Z) letters, numbers (0 to 9), or underscores (_).
- Identifiers cannot begin with a number.
 - 1variable (x)
 - variable1 (o)
- No keywords can be used as identifiers.
- No special symbols such as !, @, #, \$, % can be used as identifiers.
- Identifiers can be of any length.
- Case sensitive
 - Variable ≠ variable

Structure of the python program

Comment

A comment in programming is a detailed description of the code within a program that does not affect the program's execution.

- Comment types and lab exercise

- Block comments
 - Use three single quotation marks ('') or double quotation marks ("") in a row
- Inline comments
 - Use a number sign (#)
- Code example - Comment.py

```
"""
File name : Comment.py
Author : John
Date : x month x day in 2022
Descriptions : Commenting practice in Python3
"""

# print 'Hello World'
print('Hello World')
```

Structure of the python program

Indent

When you write code, you use indentation for readability. In Python, instead of using {} to denote spaces, the language uses indentation for this purpose. That's why indentation is a grammatical requirement in Python.

- Understanding indentation
 - Spaces and tabs cannot be mixed (automatically converted when working in IDLE, PyCharm).
 - Organize code blocks based on indentation
 - A code block is a collection of code for a specific action.
 - The line followed by the ':' symbol (colon) after if, for, def, etc. must be indented.
 - Must have the same number of indentation spaces within the same block.
 - Otherwise an "IndentationError: unexpected indent" will occur.
 - Indentation shortcuts in PyCharm (after setting up blocks where needed)
 - Tab : move block right indented
 - Shift + Tab : move block left indented

Structure of the python program

Indentation

When you write code, you use indentation for readability. In Python, instead of using {} to denote spaces, the language uses indentation for this purpose. That's why indentation is a grammatical requirement in Python.

- Indentation lab exercise
 - Code example 1 - Indent1.py

```
"""
File name : Indent1.py
"""

# Check for indentation errors
print('Hello') # line 0
    print('World') # 2 lines
```



```
# Fix indentation errors
print('Hello')
print('World')
```

- Code example 2 - Indent2.py

```
"""
File name : Indent2.py
"""

# Check for indentation errors after the colon (:)
TestNumber = 1234
if TestNumber == 1234:
    print('TestNumber is .. ') # Line 2
        print('1234! ') # 4 lines
```



```
# Fix an indentation error after a colon (:)
TestNumber = 1234
if TestNumber == 1234:
    print('TestNumber is .. ')
    print('1234! ')
```

Structure of the python program

Creating and separating rows

Many languages use a semicolon (;) at the end of a sentence to indicate the end of a phrase. Python doesn't require a semicolon to end a statement; it simply moves to the next line. It automatically recognizes the end of a delimiter when the line changes.

- Understanding line creation with a lab exercise
 - No need to use a semicolon (;) to mark the end of a sentence
 - When writing multiple sentences on a single line, separate them with a semicolon (;).
 - Code example - Semicolon.py

```
"""
File name : Semicolon.py
"""

# Before combining rows
a = 1
b = 2
c = 3

print(a)
print(b)
print(c)

# After combining rows
a = 1; b = 2; c = 3

print(a); print(b); print(c)
```

Structure of the python program

Creating and separating rows

If the content of line 1 is long and needs to be typed on multiple lines, use a backslash (\) or parentheses.

- Understanding line breaks with a lab exercise
 - Use a backslash (\) at the end of a line to break up a sentence into lines
 - You can omit the backslash (\) after a comma (,) inside parentheses (parentheses, braces, brackets).
 - Reflected automatically working in IDLE, PyCharm
 - Code example - Backslash.py

```
"""
File name : Backslash.py
"""

# Before row separation
Twenty = 5 + 5 + 5 + 5 + 5
Forty = (10 + 10 + 10 + 10 + 10)

print(Twenty, Forty)
```

```
# After row separation
Twenty = 5 + 5 \
        + 5 + 5

Forty = (10 + 10 +
          10 + 10)

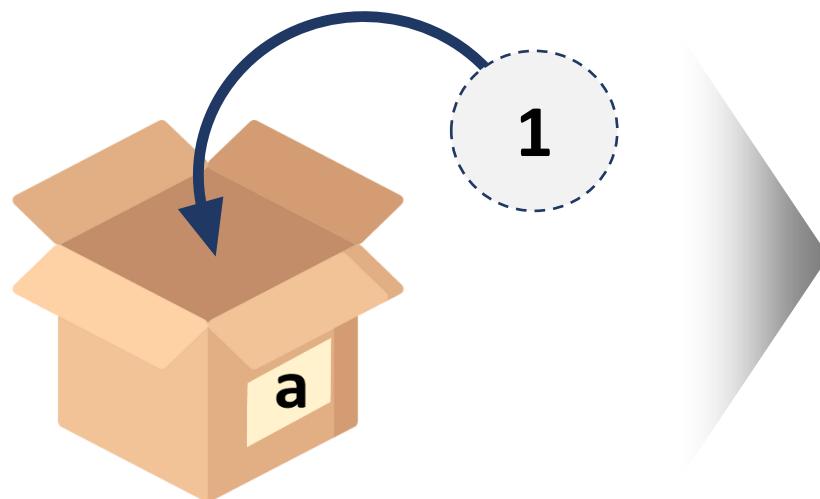
print(Twenty,
      Forty)
```

Python - variables and data types

Variable

A variable in Python is simple compared to other languages that require you to declare its type up front. In traditional languages like C, they must be strictly typed and managed (such as integer, float, character, and string).

- Understanding variables
 - As it is written, a variable is a place that can be changed.
 - It works as a container, holding values like letters or numbers.



Python - variables and data types

Variable

A variable in Python is simple compared to other languages that require you to declare its type up front. In traditional languages like C, they must be strictly typed and managed (such as integer, float, character, and string).

- Variable lab exercise
 - How to write variables
 - Equal sign (=) : means to save the value of the variable
 - Variable name : written to the left of the equal sign
 - Value : written to the right of the equal sign
 - Code example - Variable.py

```
a = 123456789
print(a)

b = 'String'
print(b)
```

<Run result>

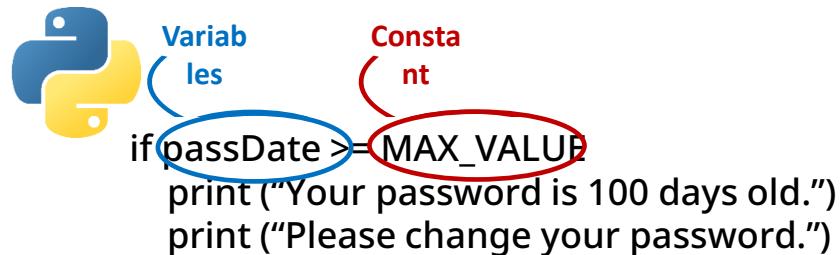
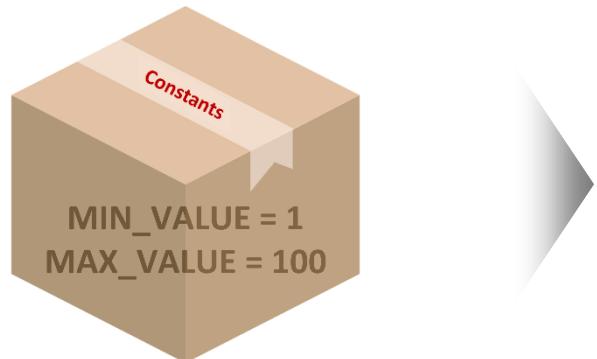
```
123456789
String
```

Python - variables and data types

Constant

A constant is a place where the same value is always stored, and they are often used to change the value of a complex number into a recognizable character.

- Understanding the constant concept
 - A place to store a value, just as a variable, but one that doesn't change.
 - E.g., Circumference (3.14), G-forces (9.8), fixed minimum and maximum values to use in code, etc.
 - Python has neither default syntax nor constants, but they can be implemented using modules, etc.
 - C.f., In C, you can specify constants directly using the const keyword.
 - Use uppercase letters and _ when explicitly assigning fixed values to variables (so that you can tell in code that the value can change, but that it is a fixed value).



Python - variables and data types

Data types overview

In any programming language, there's a saying that "if you know and understand the language's data types, you're already halfway there." We will take a closer look at the different types of data.

- What is a data type?
 - Anything you used in the form of data, such as numbers, strings, etc. during programming
 - To find out the material type of a variable or value, use the type command.
 - Output type for type commands
 - Integer : <class 'int'>
 - Float : <class 'float'>
 - Complex : <class 'complex'>
 - String : <class 'str'>
 - Tuple : <class 'tuple'>
 - List : <class 'list'>
 - Dictionary : <class 'dict'>.
 - Boolean : <class 'bool'>

Python - variables and data types

Data types overview

Every value in Python has a data type. Data types include numeric, string, list, tuple, dictionary, set, and bool.

- Data type output lab exercise
 - Code Example - Datatype.py

```
number1 = 12345
number2 = 3.14
number3 = 5.5 + 3.1j
string1 = 'test'
tuple1 = (1, 2, 3, 4, 5)
list1 = [1, 2, 3, 4, 5]
dict1 = {1, 2, 3, 4, 5}
bool1 = True

print(f"{number1} =", type(number1))
print(f"{number2} =", type(number2))
print(f"{number3} =", type(number3))
print(f"{string1} =", type(string1))
print(f"{tuple1} =", type(tuple1))
print(f"{list1} =", type(list1))
print(f"{dict1} =", type(dict1))
print(f"{bool1} =", type(bool1))
```

<Run result>

```
12345 = <class 'int'>
3.14 = <class 'float'>
(5.5+3.1j) = <class 'complex'>
test = <class 'str'>
(1, 2, 3, 4, 5) = <class 'tuple'>
[1, 2, 3, 4, 5] = <class 'list'>
{1, 2, 3, 4, 5} = <class 'set'>
True = <class 'bool'>
```

Python - variables and data types

String

A numeric type is a data type consisting of numbers, including integers, real, and complex.

- Numeric data types
 - Integer
 - Positive integers, zeros, and negative integers
 - Expressed as 0b for binary, 0o for octal, and 0x for hexadecimal
 - Floating
 - Numbers with decimal places
 - Complex
 - $x + yj$, where x is the real part and y is the imaginary part.
 - Allow you to separate real numbers from imaginary numbers

Python - variables and data types

String

A string is a collection of characters that can be made up of letters, words, etc.

- What is a string?
 - A set of characters consisting of letters, words, etc.
 - The string encoding in Python3.x is UTF-8 (2.x: Unicode).
 - When numbers are enclosed in quotes, they are treated as strings.
- How to express strings
 - Wrap them in single or double quotation marks
 - Wrap them with 3 single or double quotation marks in a row

```
print("First") # Wrap it in a pair of double quotation marks  
  
print('Second') # Wrap in a pair of single quotation marks  
  
print("""Third""") # Wrap in three double quotation marks  
  
print('''Fourth''') # Wrap in three single quotation marks
```

Python - variables and data types

String

A string is a collection of characters that can be made up of letters, words, etc.

- How to express strings
 - Wrap them in single or double quotation marks
 - Wrap them with 3 single or double quotation marks in a row

```
print("First") # Wrap it in a pair of double quotation marks  
  
print('Second') # Wrap in a pair of single quotation marks  
  
print("""Third""") # Wrap in three double quotation marks  
  
print('''Fourth''') # Wrap in three single quotation marks
```

Python - variables and data types

String

A string is a collection of characters that can be made up of letters, words, etc.

- Different ways to express strings
 - See the example below when you want to assign a multiline string to a variable :

```
# Insert escape code \n to wrap a line
multiline1 = "Life is too short\nYou need python"

# Use three consecutive single quotation marks ('') or three double quotation marks ("")
Multiline2 = '''
Life is too short
You need python
'''

print(multiline1)
print(multiline2)
```

Python - variables and data types

String

In Python, you can add or multiply strings. This is a feature that is not easily found in other languages, and is one of Python's strengths.

- String operations

- Concatenate strings by adding them together

```
head = "Python"  
tail = " is fun!"  
  
print(head + tail)
```

<Run result>

Python is fun!

- Multiply strings

```
a = "python"  
  
print(a * 3)
```

<Run result>

pythonpythonpython

- Practice question : utilize string addition and multiplication to output the following example.

[Output example]
=====

My Program
=====

Python - variables and data types

String

In Python, you can get the length of a string using the built-in len function.

- String operations
 - Find the length of a string
 - Use the len function
 - The len function is a default built-in function in Python and can be used that can be used without configuration.

```
a = "python is very funny!"  
print(len(a))
```

<Run result>

21

Python - variables and data types

String

Indexing means to "point" to something, and slicing means to "cut" something.

- Indexing

- Means to point to something
- Number each character of a string stored in a variable sequentially from zero
- Number in reverse order from -1 to read the string from the back

```
string = "Hello World!"
```

```
print(string[10])  
print(string[-2])
```

<Run result>

```
d  
d
```

H	e	I	I	o			W	o	r	I	d	!
0	1	2	3	4	5	6	7	8	9	10	11	
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

Diagram illustrating string indexing for "Hello World!". The string is represented as a sequence of characters in a grid. The first row shows the characters H, e, I, I, o, (empty), W, o, r, I, d, !. The second row shows their standard indices: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11. The third row shows their negative indices: -12, -11, -10, -9, -8, -7, -6, -5, -4, -3, -2, -1. Arrows point from the labels string[0] and string[1] to the first two characters in the string. Another set of arrows points from the labels string[10] and string[11] to the last two characters in the string.

Python - variables and data types

String

Indexing means to "point" to something, and slicing means to "cut" something.

- Slicing

- Means to cut something off
- Can extract words, not just single characters
- For $a[x : y]$, extract up to $a[x], a[x+1], a[x+2], \dots, a[y-2], a[y-1]$
- If the start or end number is omitted, the string is extracted from the beginning or end of the string.

```
string = "Hello World!"
```

```
print(string[0:5])
print(string[-12:-7])
print(string[:5])
print(string[6:])
```

<Run result>

```
Hello
Hello
Hello
World!
```

H	e	I	I	o			w	o	r	I	d	!
0	1	2	3	4	5	6	7	8	9	10	11	
-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

Python - variables and data types

String

Indexing means to "point" to something, and slicing means to "cut" something.

- Slicing
 - Example
 - Print slicing by year, month, day, and hour

```
a = '2020040615:30'  
Year = a[:4]  
Month = a[4:6]  
Day = a[6:8]  
Time = a[8:]  
print(f"{Year}Year {Month}Month {Day}Day Time: {Time}")
```

Python - variables and data types

String

The string data type has its own set of functions. These functions are also called string built-in functions. To use these built-in functions, you just need to write a string variable name followed by a "." and then the function name.

- String-related functions

Functions	Description
<code>count()</code>	<ul style="list-style-type: none">• Count the number of characters
<code>find()</code>	<ul style="list-style-type: none">• Find where characters are
<code>index()</code>	<ul style="list-style-type: none">• Find where characters are
<code>join()</code>	<ul style="list-style-type: none">• Insert string
<code>upper()</code>	<ul style="list-style-type: none">• Replace lowercase with uppercase
<code>lower()</code>	<ul style="list-style-type: none">• Replace uppercase with lowercase
<code>lstrip()</code>	<ul style="list-style-type: none">• Clear left space
<code>rstrip()</code>	<ul style="list-style-type: none">• Clear right space
<code>strip()</code>	<ul style="list-style-type: none">• Clear both spaces
<code>replace()</code>	<ul style="list-style-type: none">• Replace string
<code>split()</code>	<ul style="list-style-type: none">• Split string

Python - variables and data types

String

- String-related functions
 - count function : return the number of specified characters in a string
 - find function : return the first occurrence of the specified character in a string, or -1 if the character or string does not exist

```
# Count the number of characters using count()
a = "apple"
result1 = a.count('p')

print('a.count :', result1)

# Find where characters are using find()
a = "Python is the best choice"
result2 = a.find('b')
result3 = a.find('k')

print(f'b : {result2} / k : {result3}')
```

<Run result>

```
a.count : 2
b: 14 / k : -1
```

Python - variables and data types

String

- String-related functions
 - index function : return the first occurrence of the specified character in a string, returning an error if it does not exist

```
# Find where characters are using index()
>>> a = "Apple is very delicious"
>>> a.index('v')
9
>>> a.index('k')
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    a.index('k')
ValueError: substring not found
```

IDLE Shell

Python - variables and data types

String

The join function accepts not only strings as input, but also lists and tuples, which we will learn about later.

- String-related functions

- join function : insert a specified character between each character of a string
- upper and lower functions : letter case conversion

```
# Insert string
string = 'abcd'
print(", ".join(string))

# Change to uppercase
lalpha = 'hi'
print(lalpha.upper())

# change to lowercase
ualpha = 'HI'
print(ualpha.lower())
```

<Run result>

```
a,b,c,d
HI
hi
```

Python - variables and data types

String

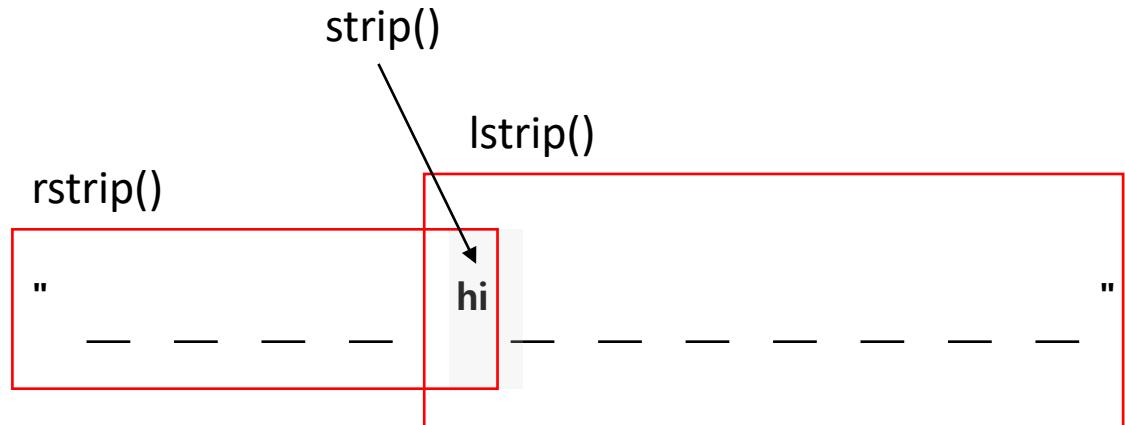
- String-related functions
 - lstrip function : remove all consecutive spaces from one or more leftmost characters from a string
 - rstrip function : remove all rightmost one or more consecutive spaces from a string
 - strip function: remove all consecutive spaces on both sides of a string.

```
# Clear left space using lstrip()
>>> a = " hi "
>>> a.lstrip()
"hi "

# Remove right space using rstrip()
>>> a= " hi "
>>> a.rstrip()
" hi"

# Remove space on both sides using strip()
>>> a = " hi "
>>> a.strip()
"hi"
```

IDLE Shell



Python - variables and data types

String

- String-related functions
 - replace function : replace a given value in a string with another value
 - split function : if you don't specify any value, it will split the string based on whitespace (space, tab, enter, etc.) and store it in a list.
 - If there is a specific value, it splits the string using the value in parentheses as the separator and stores it in a list.

```
# Replace a string using replace()
apple = "Apple is delicious"
banana = apple.replace("Apple", "Banana")

print(banana)

# split a string suing split()
string1 = "Apple is delicious"
string2 = "a:b:c:d"

result1 = string1.split()
result2 = string2.split(":")

print(result1)
print(result2)
```

<Run result>

```
Banana is delicious
['Apple', 'is', 'delicious']
['a', 'b', 'c', 'd']
```

Python - variables and data types

Bool

A bool is a data type that is an expression of either True or False.

- What is a bool (Boolean) data type?
 - A data type that express what is True and False
 - True and False are Python reserved words, not strings, so the first character is always uppercase.
 - Use as a return value for conditional statements
 - Becomes False if
 - Strings, lists, tuples, dictionaries, etc. are empty without values such " ", [], (), { }
 - None
 - 0

Python - variables and data types

Bool

A bool is a data type that is an expression of either True or False.

- Bool
 - Bool example

```
>>> a = True
>>> b = False

>>> type(a)
<class 'bool'>
>>> type(b)
<class 'bool'>

# Return value of the conditional statement
>>> 1 == 1
True
>>> 2 > 1
True
>>> 2 < 1
False
```

Python - variables and data types

Bool

A bool is also a built-in function that allows you to identify True and False for a data type.

- bool operations
 - Use bool functions to identify True and False for a data type

```
>>> bool('acs')  
True
```

```
>>> bool('')  
False
```

```
>>> bool([1,2,3])  
True
```

```
>>> bool([])  
False
```

```
>>> bool(0)  
False
```

```
>>> bool(3)  
True
```

Python - variables and data types

Tuple

A tuple is almost identical to a list, with the following differences: lists are enclosed in [], but tuples are enclosed in (). Lists can create, delete, and modify element values, while tuples can't change their values.

- What is a tuple?
 - Collection of data
 - Element values in a tuple cannot be changed or deleted.
 - Different ways to write tuples
 - List data with commas (,) or with commas in parentheses ()
 - To create a tuple of length 1, you must do so with a comma (,) at the end
 - Can create with other data types
 - Can write tuples inside tuples

Python - variables and data types

Tuple

A tuple is almost identical to a list, with the following differences: lists are enclosed in [], but tuples are enclosed in (). Lists can create, delete, and modify element values, while tuples can't change their values.

- Example of writing a tuple

```
# List data with commas (,) or commas in parentheses ()
tuple1 = ()
tuple2 = 1, 2, 3
tuple3 = (1, 2, 3)

# Must be followed by a comma (,) to create a tuple of length 1
tuple4 = (1,)

# Can create with other data types
tuple5 = 'acs', 123, True

# Can create a tuple inside a tuple
tuple6 = ('a', 'b', ('ab', 'cd'))
```

Python - variables and data types

Tuple

A tuple is almost identical to a list, with the following differences: lists are enclosed in [], but tuples are enclosed in (). Lists can create, delete, and modify element values, while tuples can't change their values.

- Element values in a tuple cannot be changed or deleted.
 - Element values cannot be changed.

```
# Element values cannot be changed.
```

```
>>> t = (1,2,3)
>>> t[0]=4
```

[Result]
Traceback (most recent call last):
 File "<pyshell#3>", line 1, in <module>
 t[0]=4
TypeError: 'tuple' object does not support item assignment

- Cannot delete element values

```
# Element values cannot be deleted.
```

```
>>> t = (1,2,3)
>>> del t[1]
```

[Result]
Traceback (most recent call last):
 File "<pyshell#2>", line 1, in <module>
 del t[1]
TypeError: 'tuple' object doesn't support item deletion

Python - variables and data types

Tuple

Indexing is the sequential numbering from 0 to each character of a string stored in a variable. To read the string backwards, it is numbered in reverse order from -1.

- Tuple indexing and slicing
 - Tuple indexing
 - Example

```
>>> a = (1, 2, 'a')
>>> a[0]
1
>>> a[2]
'a'
```

Index	0	1	2
Element value	1	2	'a'
	-3	-2	-1

↓ ↓ ↓

a[0] a[1] a[2]

Python - variables and data types

Tuple

Slicing allows you to extract words rather than just single characters: for $a[x : y]$, it will extract $a[x], a[x+1], a[x+2], \dots, a[y-2], a[y-1]$, and if the start or end number is omitted, it will extract from the start or end of the string.

- Tuple indexing and slicing
 - Tuple slicing
 - Example

```
>>> a = (1, 2, 'a', 'b')
>>> a[1:]
(2, 'a', 'b')
```

Index	0	1	2	3
Element value	1	2	'a'	'b'
	-5	-4	-3	-2

→

$a[1:]$

Python - variables and data types

Tuple

Tuples can be combined with the + operation and repeated with the * operation. You can find the number of elements with the len function.

- Tuple operations

- Add tuples (+)

```
>>> a = (1, 2, 'a', 'b')
>>> b = (3, 4)
>>> a + b
(1, 2, 'a', 'b', 3, 4)
```

- Repeat a tuple (*)

```
>>> a = (3, 4)
>>> a * 3
(3, 4, 3, 4, 3, 4, 3, 4)
```

- Find the length of a tuple (len)

```
>>> a = (1, 2, 'a', 'b')
>>> len(a)
4
```

Python - variables and data types

List

A list is a collection of data. The order in which data is entered is maintained.

- What is a list?
 - Collection of data
 - Preserve typed order
 - Unlike a set, you can also create a list with a count of 1
 - Store without data type restrictions
 - How to create a list
 - list() or []
 - Each element value is separated by a comma (,)

```
ListName = [element1, element2, element3, - - - - ]
```

Python - variables and data types

List

The list can contain any data type.

- How to represent lists

- Empty list
- A list whose element values are numbers
- A list whose element values are strings
- A list containing a mixture of numbers and strings
- A list whose element values include lists

```
# Empty list
>>> a = []
>>> a= list()

# A list whose element values are numbers
>>> b = [1, 2, 3]

# A list whose element values are strings
>>> c = ['Python', 'is', 'very', 'easy']

# A list containing a mixture of numbers and strings
>>> d = [1, 2, 'Python', 'is']

# A list whose element values include lists
>>> e = [1, 2, ['Python', 'is']]
```

Python - variables and data types

List

The list can contain any data type.

- How to express a list

- Source file - "List.py"

```
list1 = list()
print(type(list1), list1)

list2 = []
print(type(list2), list2)

list3 = [1, 2, 3, 4, 5]
print(type(list3), list3)

list4 = [1]
print(type(list4), list4)

list5 = [1, 2, 3, 1.1, 2.2, 'A', True, (1, 2, 3)]
print(type(list5), list5)
```

<Run result>

```
<class 'list'> []
<class 'list'> []
<class 'list'> [1, 2, 3, 4, 5]
<class 'list'> [1]
<class 'list'> [1, 2, 3, 1.1, 2.2, 'A', True, (1, 2, 3)]
```

Python - variables and data types

List

Indexing is the sequential numbering from 0 to each character of a string stored in a variable. To read the string backwards, it is numbered in reverse order from -1.

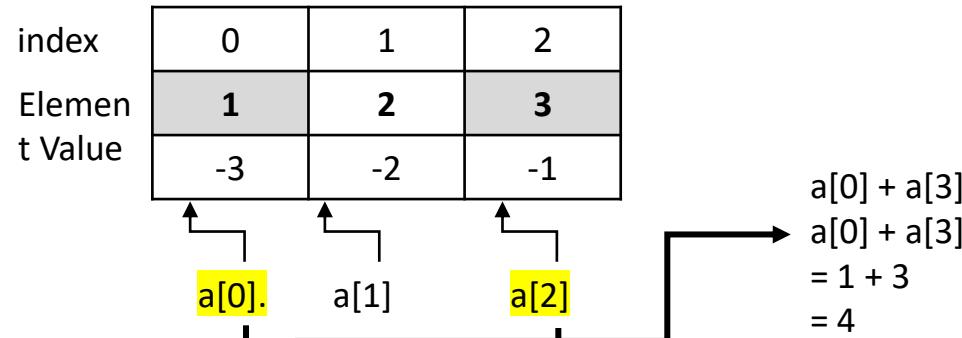
- List indexing and slicing
 - List indexing
 - Example
 - List indexing with numeric element values

```
>>> a = [1, 2, 3]
>>> a
[1, 2, 3]

>>> a[0]
1

>>> a[-1]
3

>>> a[0] + a[2]
4
```



Python - variables and data types

List

Indexing is the sequential numbering from 0 to each character of a string stored in a variable. To read the string backwards, it is numbered in reverse order from -1.

- List indexing and slicing
 - List indexing
 - Example
 - Nested list indexing

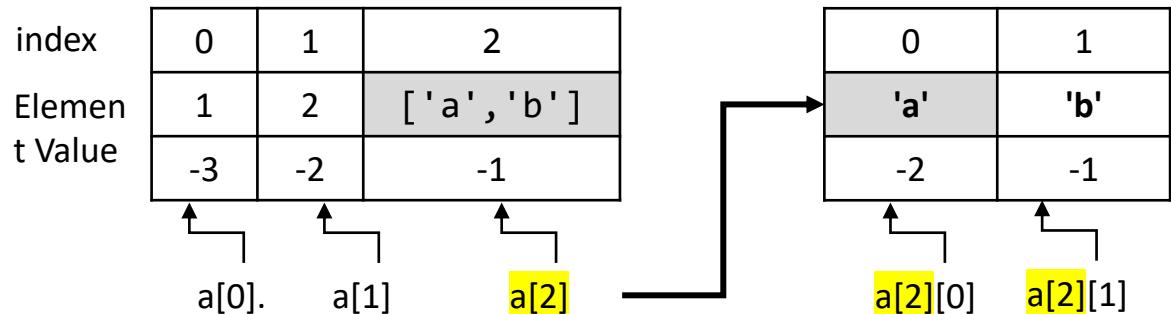
```
>>> a = [1, 2, ['a', 'b']]
```

```
>>> a[2]  
['a', 'b']
```

```
>>> a[2][0]  
'a'
```

```
>>> a[-1]  
['a', 'b']
```

```
>>> a[-1][0]  
'a'
```



Python - variables and data types

List

Indexing is the sequential numbering from 0 to each character of a string stored in a variable. To read the string backwards, it is numbered in reverse order from -1.

- List indexing and slicing

- List indexing

- Example

- Triple list indexing

```
>>> a = [1, 2, ['a', 'b', ['c', 'd']]]

>>> a[2][2][0]
'c'
```

0	1	2
1	2	['a', 'b', ['c', 'd']]
-3	-2	-1

a[0]. a[1] a[2]

0	1	2
'a'	'b'	['c', 'd']
-3	-2	-1

a[2][0] a[2][1] a[2][2]

0	1
'c'	'd'
-2	-1

a[2][2][0] a[2][2][1]

Python - variables and data types

List

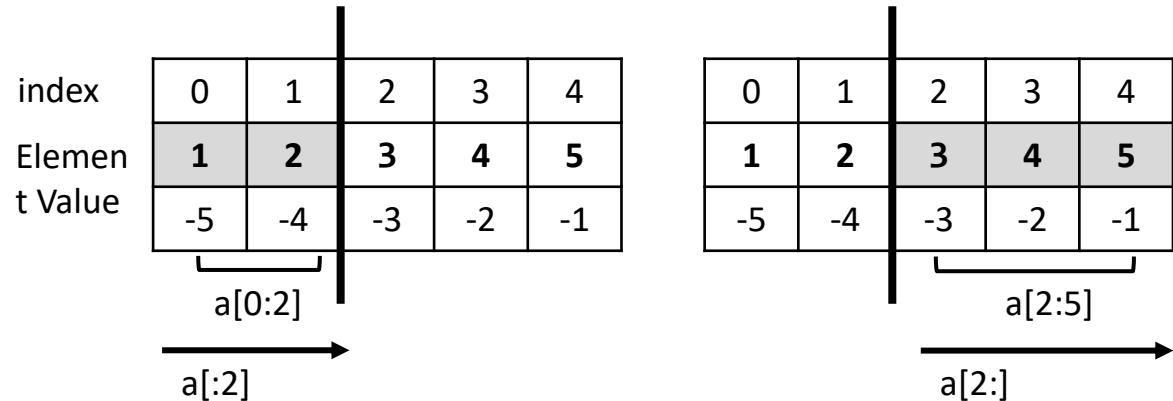
Slicing can extract words rather than just single characters: for $a[x : y]$, it will extract $a[x], a[x+1], a[x+2], \dots, a[y-2], a[y-1]$, and if the start or end number is omitted, it will extract from the beginning or end of the string.

- Indexing and slicing lists
 - List slicing
 - Example
 - Slicing a list with numeric element values

```
>>> a = [1, 2, 3, 4, 5]
>>> a[0:2]
[1, 2]

>>> a[:2]
[1, 2]

>>> c = a[2:]
[3, 4, 5]
```



Python - variables and data types

List

Slicing allows you to extract words rather than just single characters: for $a[x : y]$, it will extract $a[x], a[x+1], a[x+2], \dots, a[y-2], a[y-1]$, and if the start or end number is omitted, it will extract from the start or end of the string.

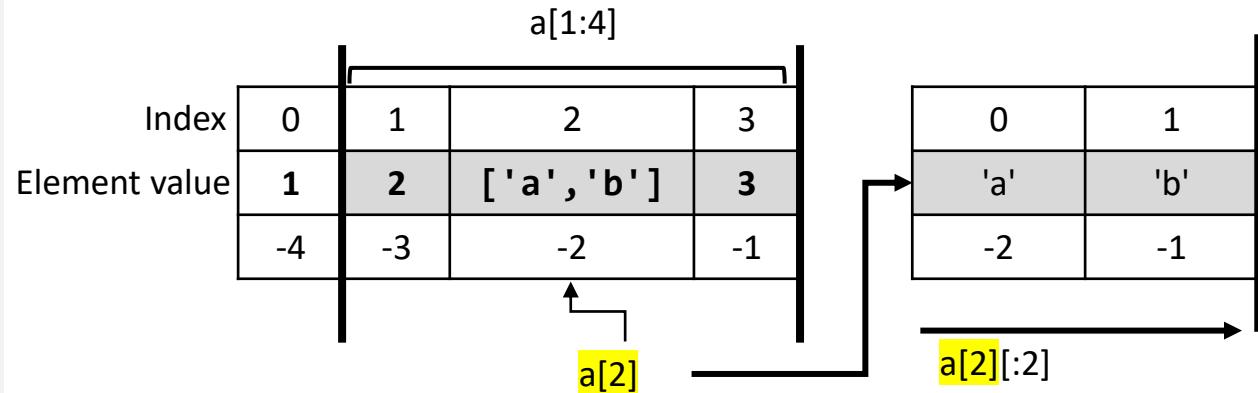
- List indexing and slicing

- List slicing

- Example

- Nested list slicing

```
>>> a = [1, 2, ['a', 'b'], 3]
>>> a[1:4]
[2, ['a', 'b'], 3]
>>> a[2][:2]
['a', 'b']
```



Python - variables and data types

List

As with strings, lists can be combined with a plus (+) sign and repeated with an asterisk (*) sign. You can find the length of a string with the len function.

- List operations

- Add lists (+)

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a + b
[1, 2, 3, 4, 5, 6]
```

- Repeat a list (*)

```
>>> a = [1, 2, 3]
>>> a * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

- Find the length of a list

```
>>> a = [1, 2, 3]
>>> len(a)
3
```

Python - variables and data types

List

The del function is a built-in deletion function in Python. It deletes the value of the xth element of the list a[x].

- Edit and delete lists

- Modify values in a list

```
>>> a = [1, 2, 3]
>>> a[2] = 4
>>> a
[1, 2, 4]
```

- Delete list elements using the del function

```
>>> a = [1, 2, 3]
>>> del a[1]
>>> a
[1, 3]
```

- Delete multiple list elements with the del function

```
>>> a = [1, 2, 3, 4, 5]
>>> del a[2:]
>>> a
[1, 2]
```

Python - variables and data types

List

As with strings, you can use a number of list-related functions by prefixing the function with the name of a list, followed by a ".":

- List-related functions

Functions	Description
<code>append()</code>	<ul style="list-style-type: none">• Add elements to a list
<code>sort()</code>	<ul style="list-style-type: none">• Sort a list
<code>reverse()</code>	<ul style="list-style-type: none">• Reverse a list
<code>index()</code>	<ul style="list-style-type: none">• Return the position of a specified element in a list
<code>insert()</code>	<ul style="list-style-type: none">• Insert elements into a list
<code>remove()</code>	<ul style="list-style-type: none">• Remove list elements
<code>pop()</code>	<ul style="list-style-type: none">• Pull out list elements
<code>count()</code>	<ul style="list-style-type: none">• Count the number of elements x in a list
<code>extend()</code>	<ul style="list-style-type: none">• Expand a list

Python - variables and data types

List

As with strings, you can use a number of list-related functions by prefixing the function with the name of a list, followed by a ".":

- List-related functions
 - append function : add x to the end of the list
 - Any data type can be added to the list.

```
# Add elements to the list with append()
>>> a = [1, 2, 3]
>>> a.append(4)
>>> a
[1, 2, 3, 4]

>>> a.append([5,6])
>>> a
[1, 2, 3, 4, [5, 6]]
```

Python - variables and data types

List

As with strings, you can use a number of list-related functions by prefixing the function with the name of a list, followed by a ".":

- List-related functions

- pop function() : return the last element of a list and delete it from the list
- pop function(x) : return the xth element of a list and delete it from the list

```
# Sort the list with sort()
>>> a = [1, 4, 3, 2]
>>> a.sort()
>>> a
[1, 2, 3, 4]

>>> a = ['a', 'c', 'b']
>>> a.sort()
>>> a
['a', 'b', 'c']
```

Python - variables and data types

List

As with strings, you can use a number of list-related functions by prefixing the function with the name of a list, followed by a ".":

- List-related functions

- reverse function : sort the list in reverse order
 - Sort the list in reverse order, sending earlier items backward; this does not sort in order before reversing the order
- index(x) function : return the position of a specified value x if there is a value x in the list

```
# Reverse the list with reverse()
>>> a = ['a', 'c', 'b']
>>> a.reverse()
>>> a
['b', 'c', 'a']

# Return the position with index()
>>> a = [1,2,3]
>>> a.index(3)
2
>>> a.index(1)
0
```

Python - variables and data types

List

As with strings, you can use a number of list-related functions by prefixing the function with the name of a list, followed by a ".":

- List-related functions

- `insert(a, b)` function : insert b at position of a in the list
- `remove(x)` function : delete the first occurrence of x in the list

```
# Insert elements into a list with insert()
>>> a = [1, 2, 3]
>>> a.insert(0, 4)
>>> a
[4, 1, 2, 3]

# Remove an element from a list with remove()
>>> a = [1, 2, 3, 1, 2, 3]
>>> a.remove(3)
>>> a
[1, 2, 1, 2, 3]
```

Python - variables and data types

List

As with strings, you can use a number of list-related functions by prefixing the function with the name of a list, followed by a ".":

- List-related functions

- pop function() : return the last element of a list if values are empty and delete it from the list
- pop function(x) : return the xth element of a list and delete it from the list

```
# Remove elements from a list with pop()
>>> a = [1,2,3,4]
>>> a.pop()
3
>>> a
[1, 2, 4]

>>> a.pop(1)
2
>>> a
[1, 4]
```

Python - variables and data types

List

As with strings, you can use a number of list-related functions by prefixing the function with the name of a list, followed by a ".":

- List-related functions

- count(x) function : return the number of x in the list
- extend(x) function : add the list x to the list; only the list type can be in x.

```
# Count the number of elements x in the list with count()
>>> a = [1,2,3,1]
>>> a.count(1)
2

# Extend the list using with extend()
>>> a = [1,2,3]
>>> a.extend([4,5])
>>> a
[1, 2, 3, 4, 5]
>>> b = [6, 7]
>>> a.extend(b)
>>> a
[1, 2, 3, 4, 5, 6, 7]
```

Python - variables and data types

Dictionary

A dictionary is a data type whose elements are key:value pairs. Whether a key can be used as a key in a dictionary depends on whether the key is a mutable or an immutable value.

- What is a dictionary?

- A data type that has a "key" and a "value" as a pair
 - Keys can only be numbers, strings, or tuples whose values cannot be changed.
 - Values can be numbers, strings, tuples, lists, etc.
- If a key value is duplicated, only the last key:value pair created is the valid one.
- Hash table structure to quickly find a value by its key
- How to write a dictionary
 - Create a dictionary with {}
 - Write elements inside {}, separated by commas (,) if there are multiple elements
 - In Python, it is implemented as a "dict" class, which can be created with the dict() constructor

Python - variables and data types

Dictionary

A dictionary is a data type whose elements are key:value pairs.

- Example of building a dictionary

```
# Create a dictionary with {}
a = {}

# Write elements within {}, separated by commas (,) if multiple
b = {'name':'acs', 'age':20, 'birth': 0416}

# Can be created using the dict() constructor
c = dict(kor=87, eng=80, mat=85)
```

Key	Value
name	acs
age	20
birth	0416

Key	Value
en	87
eng	80
mat	85

Python - variables and data types

Dictionary

Keys can only be numbers, strings or tuples whose values cannot be changed, while values can be numbers, strings, tuples, lists, etc.

- Example of building a dictionary

```
>>> c = {'name':['acs','sk'], 'age':20, 'birth': ('0416','0901')}
>>> c['name']
['acs', 'sk']

>>> c['name'][1]
'sk'

>>> c['age']
20

>>> c['birth']
('0416', '0901')

>>> c['birth'][0]
'0416'
```

Key	Value
name	['acs', 'sk']
age	20
birth	('0416', '0901')

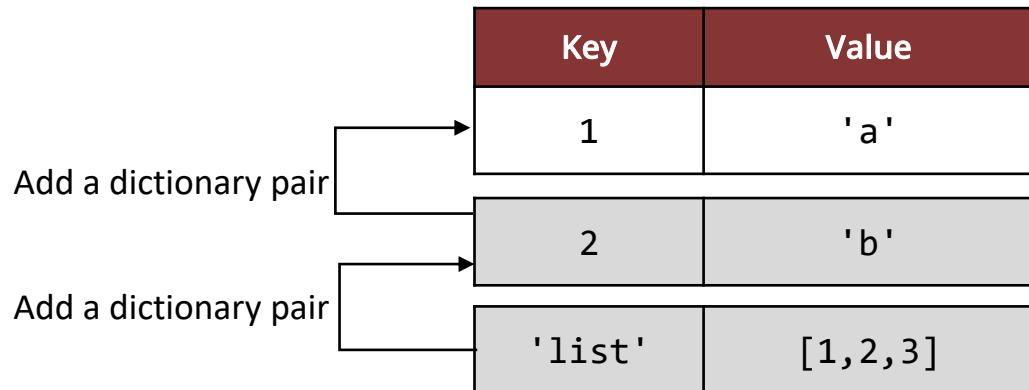
Python - variables and data types

Dictionary

Keys can only be numbers, strings or tuples whose values cannot be changed, while values can be numbers, strings, tuples, lists, etc.

- Adding and deleting dictionary pairs
 - Add pairs

```
>>> a = {1: 'a'}
>>> a[2] = 'b'
>>> a
{1: 'a', 2: 'b'}
>>> a[3] = [1,2,3]
>>> a
{1: 'a', 2: 'b', 'list': [1, 2, 3]}
```



Python - variables and data types

Dictionary

Keys can only be strings or tuples whose values cannot be changed, and values can be numbers, strings, tuples, lists, etc.

- Adding and deleting dictionary pairs
 - Delete pairs

```
>>> del a[1]
>>> a
{2: 'b', 'list': [1, 2, 3]}
```

Delete a dictionary pair



Key	Value
1	'a'
2	'b'
'list'	[1,2,3]

Python - variables and data types

Dictionary

The dictionary itself has functions to help you use the dictionary more freely.

- Dictionary-related functions

Functions	Description
<code>keys()</code>	<ul style="list-style-type: none">• Create a list of keys
<code>values()</code>	<ul style="list-style-type: none">• Create a list of values
<code>items()</code>	<ul style="list-style-type: none">• Get key:value pairs
<code>clear()</code>	<ul style="list-style-type: none">• Clear all key:value pairs
<code>get()</code>	<ul style="list-style-type: none">• Get values from their keys

Python - variables and data types

Dictionary

The dictionary itself has functions to help you use the dictionary more freely.

- Dictionary-related functions
 - `keys()` : gather only the keys of a dictionary and return them as a list
 - `values()` : gather only the values of a dictionary and return them as a list

```
# Create a list of keys with keys()
>>> a = {'name':'acs', 'age':20, 'birth': 0416}
>>> a.keys()
['name', 'age', 'birth']
```

```
# Create a list of Values with values()
>>> a.values()
['acs', 20, 0416]
```

Python - variables and data types

Dictionary

The dictionary itself has functions to help you use the dictionary more freely.

- Dictionary-related functions

- items function : return a tuple of pairs of keys and values as a list of values
- clear function : delete all elements in a dictionary

```
# Get key:value pairs with items()
>>> a = {'name':'acs','age':20, 'birth': 0416}
>>> a.items()
[('name', 'acs'), ('age', 20), ('birth', 0416)]
```

```
# Clear all key:value pairs with clear()
>>> a.clear
>>> a
{}
```

Python - variables and data types

Dictionary

The dictionary itself has functions to help you use the dictionary more freely.

- Dictionary-related functions

- get(x) function : return the value corresponding to its key x
 - Return None for non-existent keys
 - Use get(x,'default value') to create a new key x with a specified (default) value in a dictionary

```
# Get values by their keys with get()
>>> a = {'name':'acs','age':20, 'birth': 0416}
>>> a.get('name')
'acs'

>>> print(a.get('phone'))
None

>>> print(a.get('phone', 'non-existent element'))
'non-existent element'
```

Python - variables and data types

Set

A set is a data type that removes duplicates in any order and is often used as a filter. Because they cannot have order, they do not support indexing.

- What is a set data type?
 - A data type that removes all duplicates in any order
 - Often used as a filter to remove duplicates of a data type
 - Does not support indexing because there is no order
 - To access values stored in a set data type by indexing, convert them to a list or tuple and then access them
 - How to write a set
 - Write with set()
 - Write with {}

Python - variables and data types

Set

A set is a data type that removes duplicates in any order and cannot be ordered.

- Example of creating a set

```
# Write a set with set()
>>> s1 = set([1,2,3])
>>> s1
{1, 2, 3}

>>> s2 = set("Hello")
>>> s2
{'e', 'H', 'l', 'o'}

# Write a set with {}
>>> s1 = {1,2,3}
>>> s1
{1, 2, 3}
```

Python - variables and data types

Set

An & sign or the intersection function is used to find the intersection.

- Find intersection, union, and difference sets
 - Intersection
 - Use an & sign or the intersection function

```
>>> s1 = set([1, 2, 3, 4, 5, 6])
>>> s2 = set([4, 5, 6, 7, 8, 9])

>>> s1 & s2
{4, 5, 6}

>>> s1.intersection(s2)
{4, 5, 6}
```

Python - variables and data types

Set

A pipe (|) or the union function is used to find the union.

- Find intersection, union, and difference sets
 - Union
 - Use a pipe (|) or the union function

```
>>> s1 = set([1, 2, 3, 4, 5, 6])
>>> s2 = set([4, 5, 6, 7, 8, 9])

>>> s1 | s2
{1, 2, 3, 4, 5, 6, 7, 8, 9}

>>> s1.union(s2)
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Python - variables and data types

Set

A minus (-) sign or the difference function is used to find the difference.

- Find intersection, union, and difference sets
 - Difference
 - Use a minus (-) sign or the difference function

```
>>> s1 = set([1, 2, 3, 4, 5, 6])
>>> s2 = set([4, 5, 6, 7, 8, 9])

>>> s1 - s2
{1, 2, 3}
>>> s2 - s1
{8, 9, 7}

>>> s1.difference(s2)
{1, 2, 3}
>>> s2.difference(s1)
{8, 9, 7}
```

Python - variables and data types

Set

You can use a number of set-related functions by prefixing the function with the name of a set, followed by a ":".

- Set-related functions

Functions	Description
<code>add()</code>	<ul style="list-style-type: none">• Add 1 value
<code>update()</code>	<ul style="list-style-type: none">• Add multiple values
<code>remove()</code>	<ul style="list-style-type: none">• Remove a specific value

Python - variables and data types

Set

You can use a number of set-related functions by prefixing the function with the name of a set, followed by a ":".

- Set-related functions
 - Function usage examples

```
# Add 1 value with add()
>>> s1 = set([1, 2, 3])
>>> s1.add(4)
>>> s1
{1, 2, 3, 4}

# Add multiple values with update()
>>> s1 = set([1, 2, 3])
>>> s1.update([4, 5, 6])
>>> s1
{1, 2, 3, 4, 5, 6}

# Remove a specific value with remove()
>>> s1 = set([1, 2, 3])
>>> s1.remove(2)
>>> s1
{1, 3}
```

Python - operator

Operators

- What is an operator in programming?
 - An operator is a construct that directs a computer to perform a mathematical and logical process.
 - Types of operators in Python
 - Arithmetic operators +, -, *, /, //, **, and %.
 - Relational operators ==, !=, <, >, <= , >=
 - Assignment operators =, +=, -=, *=, /=, %=, //=
 - Logical operators and, or, not
 - Bitwise operators &, |, ^, ~, <<, >>
 - Membership operators in, not in
 - Identity operators is, is not
 - And so on.



Python - operator

Operators

- Arithmetic operators
 - Operators that perform numeric calculations

Operator	Description	Example	Output value
+	• Add two operands	5 + 2	7
-	• Subtract 2 operands	5 - 2	3
*	• Multiply two operands	5 * 2	10
**	• Exponentiate two operands	5 ** 2	25
/	• Divide two operands	5 / 2	2.5
//	• The value of a quotient divided by two operands (with the decimal places ignored)	5 // 2	2
%	• The remainder of the division of two operands	5% 2	1

Python - operator

Operators

- Arithmetic operators
 - Code example - Arithmetic.py

```
num1 = 5
num2 = 2

add = num1 + num2 # Addition
sub = num1 - num2 # Subtraction
mul = num1 * num2 # Multiplication
exp = num1 ** num2 # Exponentiation
div = num1 / num2 # Division
quot = num1 // num2 # Quotient
rem = num1% num2 # Remainder

print(add)
print(sub)
print(mul)
print(squ)
print(div)
print(quot)
print(rem)
```

<Run result>

```
7
3
10
25
2.5
2
1
```

Python - operator

Operators

- Relational (comparison) operators
 - Operators that compare operands to make a true or false determination.
 - True means any non-zero number, False means zero

Operator	Description	Example. (a = 10; b = 20)	Output value
<code>==</code>	<ul style="list-style-type: none">• Compare two operands to see if they are equal	<code>a == b</code>	False
<code>!=</code>	<ul style="list-style-type: none">• Compare two operands to see if they are different	<code>a != b</code>	True
<code>></code>	<ul style="list-style-type: none">• Compare if the left operand is greater than the right operand	<code>a > b</code>	False
<code><</code>	<ul style="list-style-type: none">• Compare if the left operand is less than the right operand	<code>a < b</code>	True
<code>>=</code>	<ul style="list-style-type: none">• Compare if the left operand is greater than or equal to the right operand	<code>a >= b</code>	False
<code><=</code>	<ul style="list-style-type: none">• Compare if the left operand is less than or equal to the right operand	<code>a <= b</code>	True

Python - operator

Operators

- Relational (comparison) operators
 - Code example - Relational.py

```
a = 5
b = 2

print(a == b)
print(a != b)
print(a > b)
print(a < b)
print(a >= b)
print(a <= b)
```

<Run result>

```
False
True
True
False
True
False
```

Python - operator

Operators

- Logical operators
 - Operators that perform logical operations (and, or, not)
 - True means any number other than 0, False means zero

Operators	Description	Syntax	Example when true
and	<ul style="list-style-type: none">• Operate on two operands as a logical product (AND)	<code>x and y</code>	<code>x = 1 and y = 1</code>
or	<ul style="list-style-type: none">• Operate on two operands as logical sum (OR)	<code>x or y</code>	<code>x = 1 or y = 1</code> <code>x = 1 or y = 0</code> <code>x = 0 or y = 1</code>
not	<ul style="list-style-type: none">• Operate on one operand as logical negation (NOT)	<code>Not x</code>	<code>x = 0</code>

Python - operator

Operators

- Logical operators example
 - Code example - Logical.py

```
a = 1  
b = 0  
  
print(a and b)  
print(a or b)  
print(not a)  
print(not b)
```

<Run result>

```
False  
True  
False  
True
```

Python - operator

Operators

- Ternary operators
 - Ternary operators with AND and OR
 - Ternary operators with IF and ELSE
 - In the ternary operator "a and b or c," if b is 0,
 - Even if a is True, it will eventually return c because it will be recognized as False if b goes to 0.
 - This led to the recommendation to use ternary operators with IF and ELSE.

Operator	Description
If both the condition and the condition are True or if the condition is False	<ul style="list-style-type: none">• Ternary operators with AND and OR
If the condition is True, then if the condition, else the condition is False	<ul style="list-style-type: none">• Ternary operators with IF and ELSE

Python - operator

Operators

- Ternary operators
 - Code example - Ternary.py

```
a = 10
b = 10

result1 = a == b and a - b or a + b
result2 = (a-b) if a == b else (a+b)

print(result1)
print(result2)
```

<Run result>

```
20
0
```

Python - operator

Operators

- Bitwise operators
 - Operators that perform operations on the bits of the operands
 - Bitwise operators operate on binary numbers, so it's important to understand and use decimal conversion.

Operator	Description	Example (a = 5; b = 2)	Output value
&	<ul style="list-style-type: none">• AND operation at a bit-level	a & b	0
	<ul style="list-style-type: none">• OR operations at a bit-level	a b	7
^	<ul style="list-style-type: none">• XOR operations at a bit-level	a ^ b	7
~	<ul style="list-style-type: none">• NOT operation at a bit-level	~a	-6
<<	<ul style="list-style-type: none">• Left shift operation at a bit-level	a << b	20
>>	<ul style="list-style-type: none">• Right shift operation at a bit-level	a >> b	1

Python - operator

Operators

- Assignment operators
 - Operators that assume the values of their operands or the result of an operation on them.

Operator	Description	Example syntax
=	• Assign the value of the right operand to the left operand	x = y
+=	• Add operands and assign the result to the left operand	x += y
-=	• Subtract operands and assign the result to the left operand	x -= y
*=	• Multiply operands and assign the result to the left operand	x *= y
/=	• Divide the operands and assign the quotient of the result to the left operand (real number)	x /= y
//=	• Divide the operands and assign the quotient of the result to the left operand (integer)	x //= y
%=	• Divide the operands and assign the remainder of the result to the left operand	x %= y
<<=	• Left shift operands and assign the result to the left operand	x <<= y
>>=	• Right shift operands and assign the result to the left operand	x >>= y
&=	• AND the operands and assign the result to the left operand	x >>= y
=	• OR the operands and assign the result to the left operand	x = y
^=	• XOR the operands and assign the result to the left operand	x ^= y

Python - operator

Operators

- Membership operators
 - Operators that determine inclusion

Operators	Description
<code>in</code>	<ul style="list-style-type: none">• Check for inclusion
<code>not in</code>	<ul style="list-style-type: none">• Check for not inclusion

- Code example - Membership.py

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

print(Member)
print('There is 3' if 3 in Member else 'There isn't 3')
print('There isn't 3' if 3 not in Member else 'There is 3')
```

<Run result>

There is 3
There is 3

Python - operator

Operators

- Identity operators
 - Used as operators to determine whether objects are identical
 - id() function : take an object as input and return the object's unique value (reference)
 - The returned id is a serial number that Python assigns to distinguish objects, and is numerically meaningless.

Operator	Description
is	<ul style="list-style-type: none">• Check if both operands point to the same Object or not
is not	<ul style="list-style-type: none">• Check for not inclusions

- Code example - Identity.py

```
a = 10
b = 10
c = 11

print(a, id(a))
print(b, id(b))
print(c, id(c))
print('same object' if a is b else 'different object')
print('same object' if a is c else 'different object')
print('different object' if a is not c else 'same object')
```

<Run result>

```
10 2536166550096
10 2536166550096
11 2536166550128
Same object
Different object
Different object
```

Python - data input/output

Standard input/output

Python supports the print function as a standard output function.

We will learn about the different methods available with the print function and how to use them.

- String output

- Both single (') and double (") quotation marks are acceptable.
- Use single quotation marks ('), double quotation marks ("), backslashes (\), or other symbols for output within a string
- Use " * times " to print the same string multiple times, " + " between strings when combining them
- Code example - Print1.py

```
"""
File name : Print1.py
"""

print('Hello World')
print("Hello World")
print("Hello"+"World")
print('"Hello World"')
print('"Hello World"')
print("\\"Hello World\\\"")
print('\\\'Hello World\\\'')
print('-' * 40)
```

<Run result>

```
Hello World
Hello World
Hello World
'Hello World'
"Hello World"
"Hello World"
'Hello World'
-----
-----
```

Python - data input/output

Standard input/output

Python supports the print function as a standard output function.

We will learn about the different methods available with the print function and how to use them.

- String output

- Use commas (,)
 - Use a sep parameter to put a separator between output objects (default : whitespace)
 - Print the last string with an end parameter for the character to output (default : newline)
- Code example - Print2.py

```
"""
File name : Print2.py
"""

print(1, 2, 3, 4, 5)
print(1, 2, 3, 4, 5, sep="/")
print("Name")
print("Hong Gil-dong")
print("Name", end=" : ")
print("Hong Gil-dong")
print("Hi", "I'm Gil-dong", sep="~", end="!! ")
```

<Run result>

```
1 2 3 4 5
1/2/3/4/5
Name
Hong Gil-dong
Name : Hong Gil-dong
Hi~I'm Gil-dong!!
```

Python - data input/output

Standard input/output

Python supports the print function as a standard output function.

We will learn about the different methods available with the print function and how to use them.

- Extended characters output
 - Support for default escape sequences
 - Code example - Print3.py

```
"""
File name : Print3.py
"""

print("Hello\\World") # Print backslash
print("Print!del\b\b\b") # Backspace 2 times
print("Hello\nWorld") # Newline
print("Hello\tWorld") # Tab
```

<Run result>

```
Hello\World
Print!
Hello
World
Hello World
```

Extended characters	Description
\'	<ul style="list-style-type: none">• Single quotation mark
\"	<ul style="list-style-type: none">• Double quotation mark
\\"	<ul style="list-style-type: none">• Backslash output
\b	<ul style="list-style-type: none">• Backspace
\n	<ul style="list-style-type: none">• Newline
\t	<ul style="list-style-type: none">• Tab character

Python - data input/output

Standard input/output

Python supports the print function as a standard output function.

We will learn about the different methods available with the print function and how to use them.

- Understanding formatting with a lab exercise
 - Formatting means inserting a specific value at a specific place in a string.
 - You can format the string with a mutable part by changing only that part.
 - How to use : "OutputFormat".format(data ..)
 - Code example - Print4.py

```
"""
File name : Print4.py
"""

Year = 2022
Name = 'Hong Gil-dong'
Age = 30

print("The current year is {}.".format(Year))
print("My name is {}.".format(Name))
print("Name: {} / Age: {}".format(Name, Age))
print("Name: {1} / Age: {0}".format(Age, Name))
```

<Run result>

```
The current year is 2022.
My name is Gil-dong Hong.
Name: Hong Gil-dong / Age: 30
Name: Hong Gil-dong / Age: 30
```

Python - data input/output

Standard input/output

Python supports the print function as a standard output function.

We will learn about the different methods available with the print function and how to use them.

- Formatting with f-strings
 - Formatting method added in Python 3.6 and later
 - More intuitive and easier to use than the older formatting methods
 - How to use : f"{data} .. {data}"
 - Code example - Print5.py

```
"""
File name : Print5.py
"""

Name = 'Hong Gil-dong'
Age = 30
Gender = 'Male'

print(f"Name : {Name} / Age : {Age} / Gender : {Gender}")
```



<Run Result
Name : Hong Gil-dong / Age : 30 / Gender : Male

Python - data input/output

Standard input/output

Python supports input functions as functions that accept standard input.

The `input` function accepts a string from a standard input device (keyboard).

- String input

- By default, any value entered is taken as a string (`str`) data type.
 - Input data can be assigned to a variable : `example = input()`
 - `input('string')` : output the string and take input from a standard input device (keyboard)
- Code Example - `Input1.py`

```
"""
File name : Input1.py
"""
```

```
print("Please enter a name ", end=": ")
Name = input()

print(f"Your name is '{Name}'.")
```

```
# Can be simplified
Name = input("Please enter a name : ")
print(f"Your name is '{Name}'.")
```

Python - data input/output

Standard input/output

Python supports input functions as functions that accept standard input.

The input function accepts a string from a standard input device (keyboard).

- Integer input

- Integer input requires data type conversion
- Code example - Input2.py

```
"""
File name : Input2.py
"""

# Addition test : before data type conversion
FirstNumber = input('Enter the first number : ')
SecondNumber = input('Enter the second number : ')

ResultNumber = FirstNumber + SecondNumber

# Print the result of adding strings together
print(ResultNumber)
print('1'+ '1')

# Print the result of adding integers together
print(1+1)
```

<Run result>

```
Enter the first number: 1
Enter the second number: 1
11
11
2
```

Python - data input/output

Standard input/output

Python supports input functions as functions that accept standard input.

The input function accepts a string from a standard input device (keyboard).

- Integer input (cont.)
 - Can accept input as an integer using an int() function
 - Modify the code example - Input2.py

```
"""
File name : Input2.py
"""

# Addition test: after data type conversion
FirstNumber = int(input('Enter the first number : '))
SecondNumber = int(input('Enter the second number : '))

ResultNumber = FirstNumber + SecondNumber

# Print the result of the addition
print(ResultNumber)
```

<Run result>

```
Enter the first number : 1
Enter the second number : 1
2
```

Python - data input/output

File input/output

You can omit `f.close()` because Python automatically closes objects in an open file when you exit the program. However, it is better to use `close()` to manually close the open file if you are trying to reuse a file that was opened in write mode without closing it, which will result in an error.

- Create a file object
 - Use Python's built-in `open()` function
 - Always close the opened file object with the file object's `close()` method when the task is finished.
 - Code example - `File.py`

```
"""
File name : File.py
"""

f = open("test.txt", "w")
f.close()

File path      Open mode
and name        and name
```

Open Mode	Description
r	Read mode (default)
w	Write mode, delete all files if they exist
x	Write mode, error if file exists
a	Write mode, add new content to the file end if it exists
b	Binary mode, used for writing byte-by-byte data
t	Text mode, used for recording text characters (default)
+	Read Write Mode

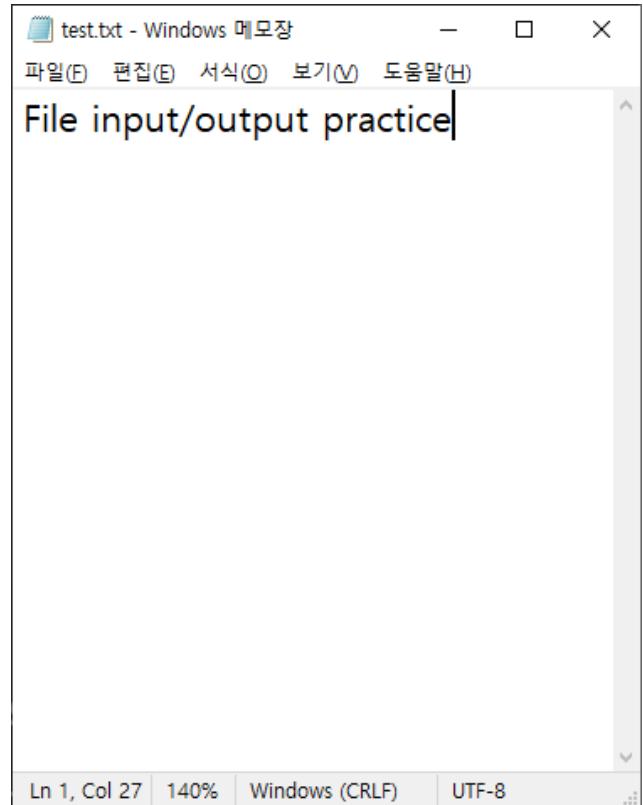
Python - data input/output

File input/output

- Write a file
 - Use a write() function
 - Modify the code example - File.py

```
"""
File name : File.py
"""

f = open("test.txt", "w")
data = "File input/output practice"
f.write(data)
f.close()
```



A screenshot of a Windows Notepad window titled "test.txt - Windows 메모장". The window contains the text "File input/output practice". The status bar at the bottom shows "Ln 1, Col 27 | 140% | Windows (CRLF) | UTF-8".

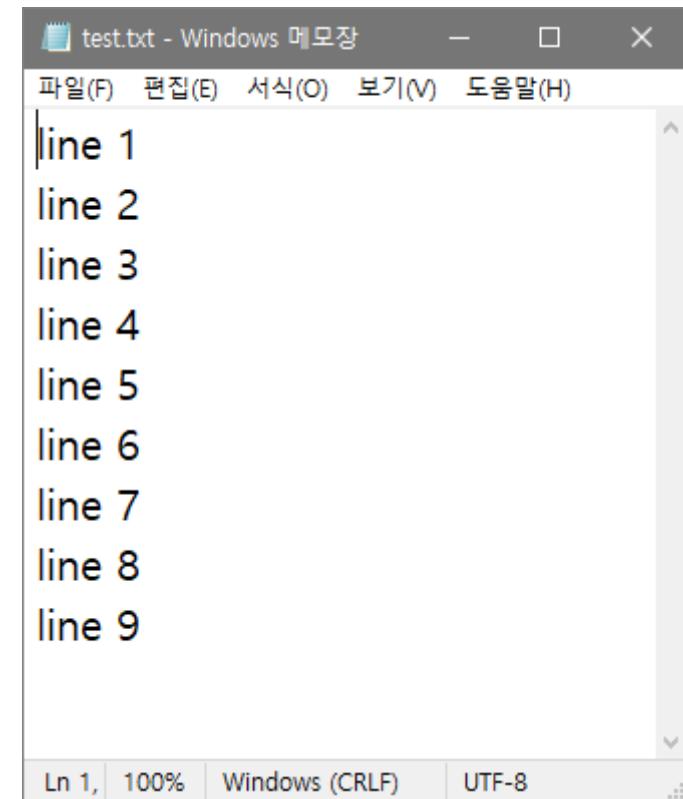
Python - data input/output

File input/output

- Write a file
 - Use a write() function
 - Modify the code example - File.py

```
"""
File name : File.py
"""

f = open("test.txt", "w")
for i in range(1, 10):
    data = f"line {i}\n"
    f.write(data)
f.close()
```



A screenshot of a Windows Notepad window titled "test.txt - Windows 메모장". The window contains the following text:

```
line 1
line 2
line 3
line 4
line 5
line 6
line 7
line 8
line 9
```

The status bar at the bottom shows "Ln 1, 100% Windows (CRLF) UTF-8".

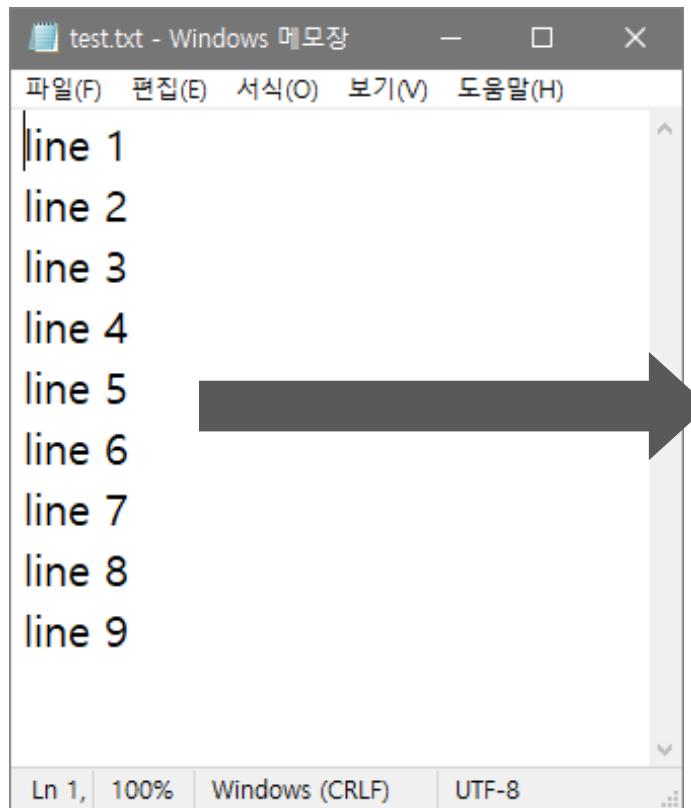
Python - data input/output

File input/output

- Read a file
 - Use a read() function : return the entire contents of the file as a string
 - Code Example - Read.py

```
"""
File name : Read.py
"""

f = open("test.txt", "r")
print(f.read())
f.close()
```



```
line 1
line 2
line 3
line 4
line 5
line 6
line 7
line 8
line 9
```

<Run result>

```
line 1
line 2
line 3
line 4
line 5
line 6
line 7
line 8
line 9
```

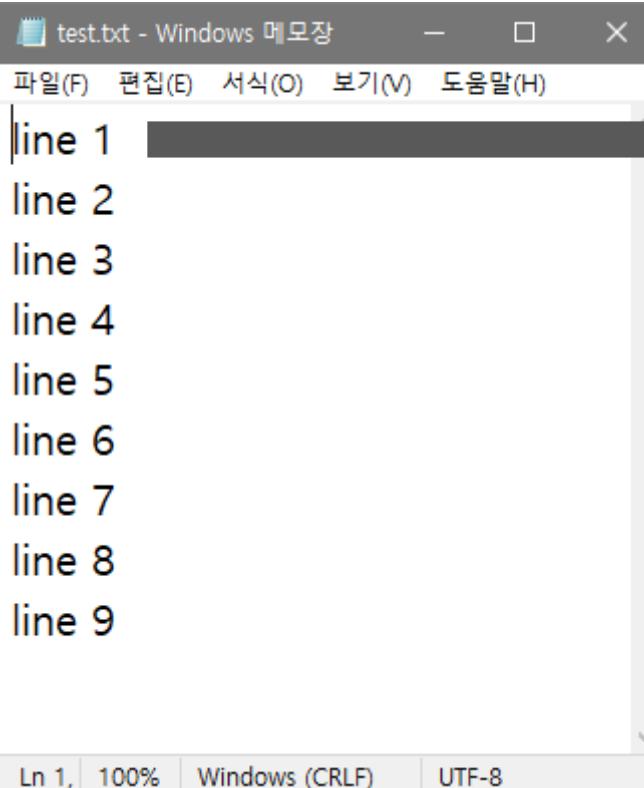
Python - data input/output

File input/output

- Read a file
 - Use a readline() function : return the first line of the file
 - Modify the code example - Read.py

```
"""
File name : Read.py
"""

f = open("test.txt", "r")
print(f.readline())
f.close()
```



A screenshot of a Windows Notepad window titled "test.txt - Windows 메모장". The window contains the following text:
line 1
line 2
line 3
line 4
line 5
line 6
line 7
line 8
line 9

<Run result>

```
line 1
```

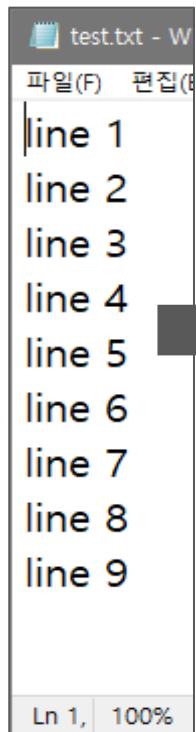
Python - data input/output

File input/output

- Read a file
 - Use a `readlines()` function : read the entire contents of the file and return it as a list that turns each line into an element.
 - Modify the code example - `Read.py`

```
"""
File name : Read.py
"""

f = open("test.txt", "r")
print(f.readlines())
f.close()
```



<Run result>

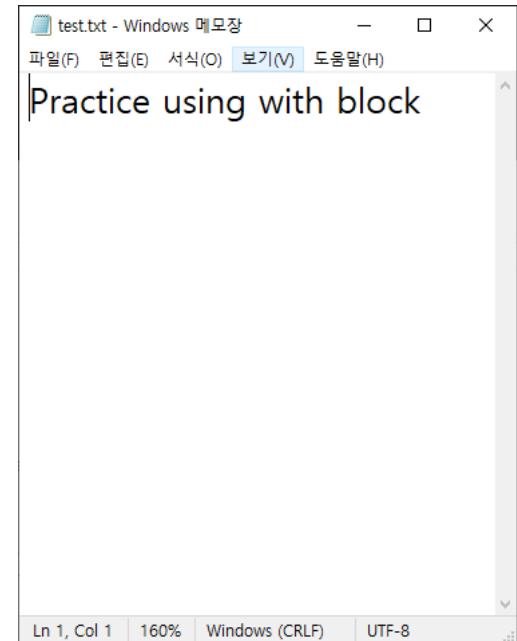
```
['line 1\n', 'line 2\n', 'line 3\n', 'line 4\n', 'line 5\n', 'line 6\n', 'line 7\n', 'line 8\n', 'line 9\n']
```

Python - data input/output

File input/output

- Use a with block
 - Close files without explicitly calling the close() method.
 - File objects are implemented internally by __enter__(), __exit__()
 - The __exit__() method is called when a with block automatically exits the block and closes the file.
 - Code example - With.py

```
....  
File name : With.py  
....  
  
with open("test.txt", "w") as f:  
    data = "Practice using with block"  
    f.write(data)
```



Python - control statements

Control statement overview

A control statement is used when you want to control the order in which each piece of code is executed in a sequential structure. There are two main types: conditional and loop.

- Control statements

- Syntax to use when you want to control the order of each piece of code executed in a sequential structure
- Always include a colon (:) at the end of a control statement
- Command to be executed within a control statement must be indented
- Types of control statements
 - Conditional statements
 - if, elif, else
 - Loops
 - for, while

Python - control statements

Types and uses of conditional statements

A conditional statement controls the flow by determining whether a particular condition is True (or a value other than zero) or False (or zero).

- Conditional statements
 - Control flow by determining whether a particular condition is true (or a value other than zero) or false (or zero)
 - Conditional statement ends where indentation ends
 - How to organize a conditional statement
 - Execute only if condition is True (if)
 - Run different code when True and False (if-else)
 - Execute when conditions are complex (if -elif -else)

Python - control statements

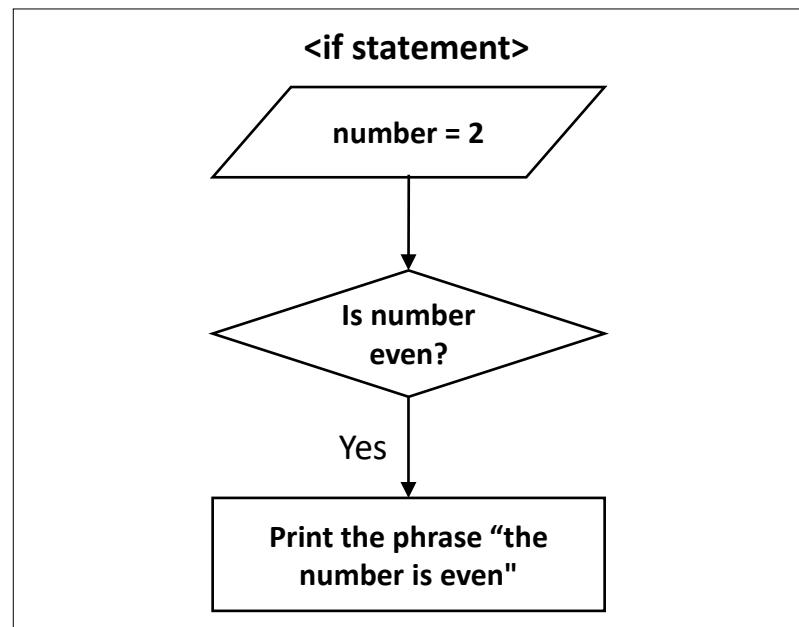
Types and uses of conditional statements

In a structure where there is only if, it will be executed if the condition is True.

- How to organize a conditional statement
 - Execute only if condition is True
 - Structure with only ifs
 - If the condition is False, nothing happens

```
if conditional statement:  
    statement to execute1  
    statement to execute2
```

...
The following command



Python - control statements

Types and uses of conditional statements

In a structure where there is only if, it will be executed if the condition is True.

- How to organize a conditional statement
 - Execute only if condition is True : example
 - Source file - "if.py"

```
# If the condition is True, the indented statement below the if statement is executed
a = 1
if a == 1:
    print('Executed because condition is true\n')

if a == 2:
    print('Not executed because condition is false\n')
print('Execute this line because it is not included in the if statement')
```

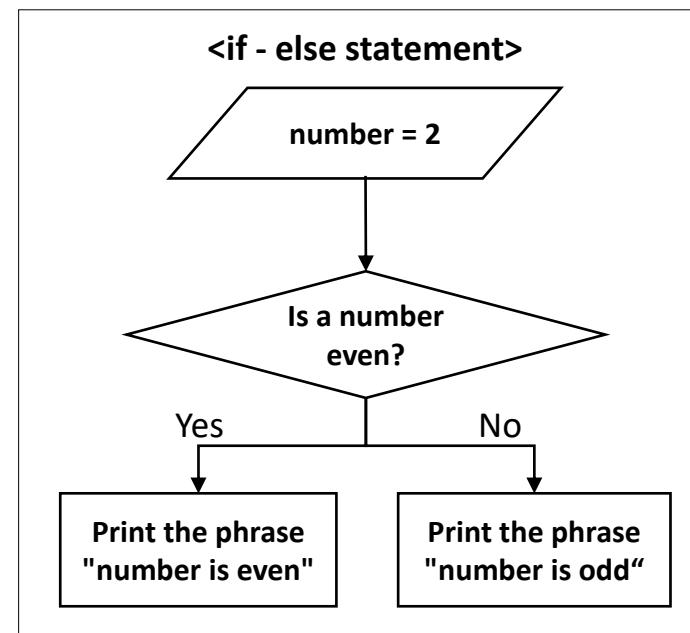
Python - control statements

Types and uses of conditional statements

A structure with both if and else executes different code on True and False, and is the most common structure.

- How to organize a conditional statement
 - Execute different code on True and False
 - Structure with ifs and elses
 - Execute the command in the if statement if the condition is True
 - Execute the command in the else statement if the condition is False

```
if conditional:  
    statement to execute1  
    statement to execute2  
  
    ...  
else:  
    statement to executeA  
    statement to executeB  
    ...
```



Python - control statements

Types and uses of conditional statements

A structure with both if and else executes different code on True and False, and is the most common structure.

- How to organize a conditional statement
 - Different code execution on True and False - example
 - Source file - "if-else.py"

```
a = 50
if a < 60:
    print("You failed because you scored {}".format(a))
else:
    print("You have a score of {}, so you passed.".format(a))

a = 80
if a < 60:
    print("You failed because you scored {}".format(a))
else:
    print("You have a score of {}, so you passed.".format(a))
```

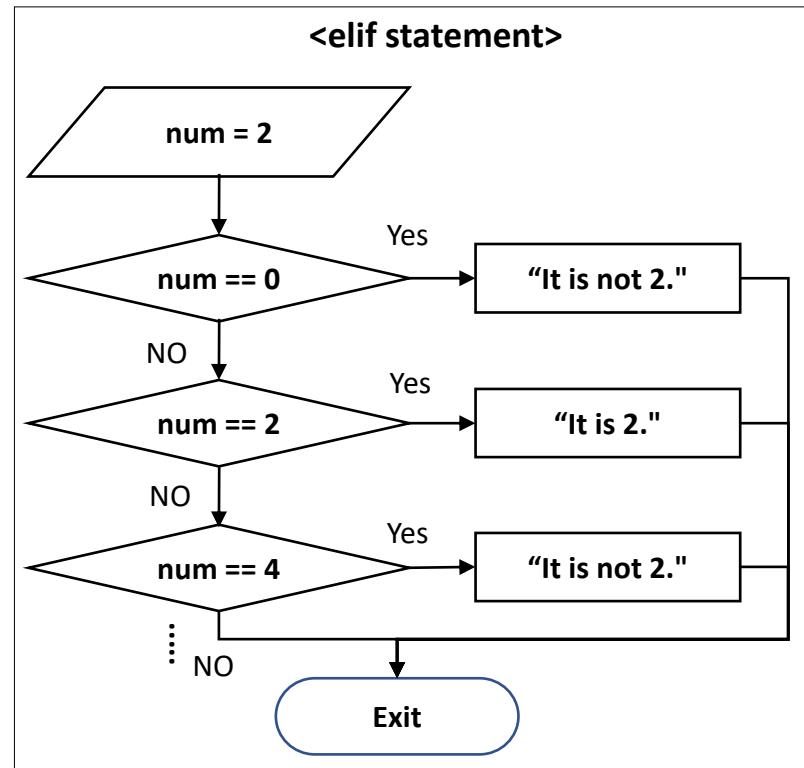
Python - control statements

Types and uses of conditional statements

A structure with if, elif, and else will be executed if the condition is complex.

- How to organize a conditional statement
 - Execute if the conditions is complex
 - Unlimited number of elifs can be used
 - Use when you have multiple conditions
 - Execute the command in the else statement if all conditions are False

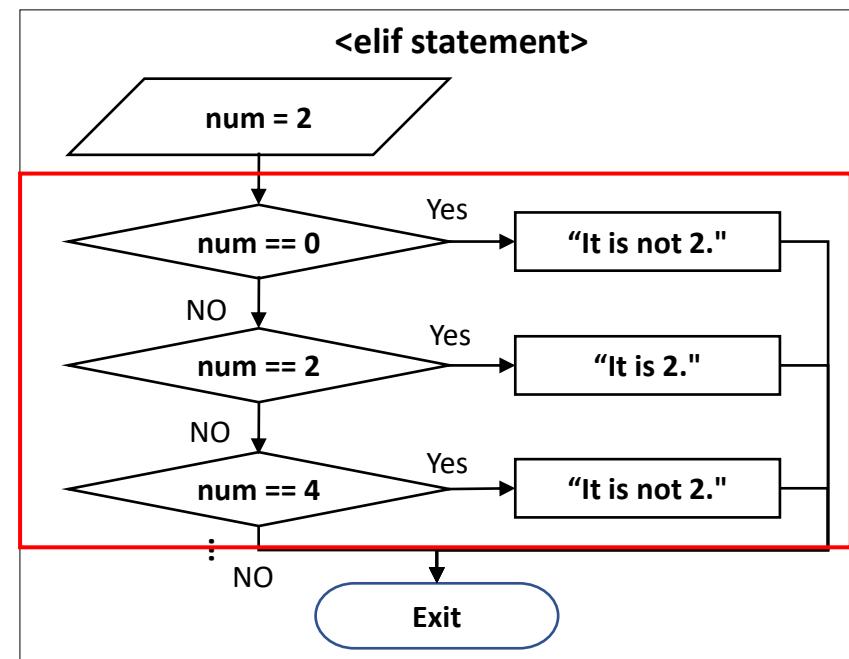
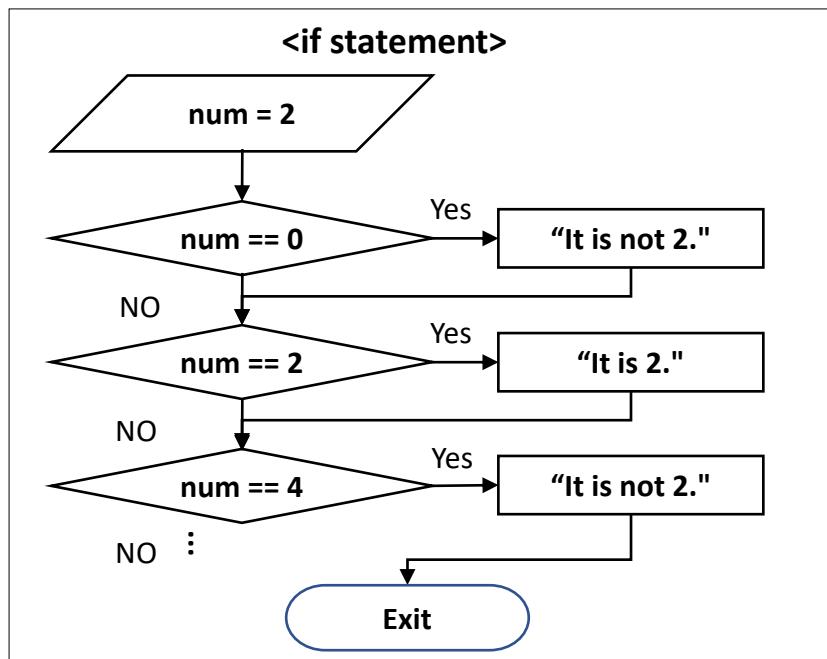
```
if <conditional statement>:  
    <Statement to execute1>  
    <statement to execute2>  
    ...  
elif <conditional statement>:  
    <statement to execute1>  
    <statement to execute2>  
    ...  
else:  
    <statement to execute1>  
    <statement to execute2>  
    ...
```



Python - control statements

Types and uses of conditional statements

- How to organize a conditional statement
 - Execute if the condition is complex
 - Use when the condition would be hard to understand and complicated if expressed only with if and else
 - Use elif to execute code after branching statements if one of the conditions is True



Python - control statements

Types and uses of conditional statements

A structure with if, elif, and else will be executed if the condition is complex.

- How to organize a conditional statement
 - Execute if the condition is complex - example
 - Source file - "if-elif-else.py"

```
num = 2
if num < 0:
    print('{} is a negative number.'.format(num))
elif num > 0:
    print('{} is a positive number.'.format(num))
else:
    print('{} is "Zero".'.format(num))
```

Python - control statements

Types and uses of loops

A loop repeats a block of code based on a given condition.

- Loops
 - Iterate over a block of code based on a given condition
 - How to organize a loop
 - while statement
 - for statement

Python - control statements

Types and uses of loops

A while statement iterates if the condition is True and terminates the iteration if the condition is False.

- How to organize a loop
 - while statement
 - Repeat if condition is True
 - Exit loop if condition is False
 - Exit the current iteration when a break command is encountered
 - When a continue command is encountered within a loop, it returns to the beginning of the current loop and compares the conditions.

```
while < conditional statement >:  
    < statement to execute1 >  
    < statement to execute2 >  
    < statement to execute3 >
```

Python - control statements

Types and uses of loops

A while statement iterates if the condition is True and terminates the iteration if the condition is False.

- How to organize a loop
 - The while statement example
 - The initial value exists outside the while statement
 - The change starting condition exists inside the while statement
 - Create an infinite loop if such condition doesn't exist inside
 - If there is a break, even if the change starting condition doesn't exist inside, the statement can avoid getting stuck in an infinite loop.
 - Source file - "while.py"

```
i = 1 # Initial value

# Repeat 10 times.
while i <= 10:
    print('{0}th '.format(i))
    i = i + 1
```

Python - control statements

Types and uses of loops

An infinite loop occurs when the conditional is True and the change starting condition does not exist inside the while statement.

- How to organize a loop
 - Infinite loop
 - It fires when the conditional statement is True and the change starting condition does not exist inside the while statement.
 - To exit an infinite loop, use the CTRL + C shortcut
 - Infinite loop example

```
# If the value of the variable is greater than 0
while True:
    # Print that string.
    print('Press CTRL + C to escape the infinite loop')
```

<Result>
Press CTRL + C to escape an infinite loop
Press CTRL + C to escape an infinite loop
Press CTRL + C to escape an infinite loop
...
Print that string over and over again

Python - control statements

Types and uses of loops

The secondary control statements for while statements are break and continue.

- How to organize a loop
 - Secondary control statements in the while statement
 - break : exit the conditional statement completely
 - Source file - "while_break.py"
 - continue : return to the conditional statement if the condition is not met
 - Source file - "while_continue.py"

```
n = 10
# Infinite loop
while True:
    n = n - 1
    print("This is {}.".format(n))

    # Call break if n goes to 0 to escape the infinite loop
    if n==0:
        print("The value of n is 0")
        break
```

```
i = 0
# While variable i is less than 10
while i < 10:
    i = i + 1
    # If a is even after dividing the variable value, call continue
    if i % 2 == 0:
        continue
    # If it's odd, escape the loop without performing the condition
    print(i)
```

Python - control statements

Types and uses of loops

A for statement is performed by assigning the first to last element of the set of data following the in operator to a variable.

- How to organize a loop
 - for statements
 - Assign the first to last element of the data set following the in operator to the variable
 - Execute "statement to execute1", "statement to execute2", etc.
 - A data set can be a string, a list, a tuple, a set, or a dictionary.
 - The else statement executes a command if there is no data in the data set and is optional

```
for variable in dataset:  
    statements to execute1  
    statements to execute2  
    ...  
else:  
    command to run when there is no data  
    ....
```

Python - control statements

Types and uses of loops

A for statement can be combined with a variety of data types.

- How to organize a loop
 - for statements example
 - Basic for statement

```
>>> test_list = ['one', 'two', 'three']
>>> for i in test_list:
...     print(i)
...
one
two
three
```

- Tuple-using for statement

```
>>> a = [(1,2), (3,4), (5,6)]
>>> for (first, last) in a:
...     print(first + last)
...
3
7
11
```

Python - control statements

Types and uses of loops

One of the secondary control statements to the for statements is the continue statement.

- How to organize a loop
 - Secondary control statements to the for statement
 - continue
 - If a continue statement is encountered while executing a command within a for statement, it returns to the beginning of the for statement.
 - Source file - "for_continue.py"

```
#Pass if score is over 60, otherwise fail
scores = [90, 25, 67, 45, 80]

for score in scores:
    if score < 60:
        print("{}Score: Fail".format(score))
        continue
    print("{}Score: Pass".format(score))
```

```
<Result>
90 points: Pass
25 points: Fail
67 points: Pass
45 points: Fail
80 points: Pass
```

Python - control statements

Types and uses of loops

Functions that are often used with for statements are range and enumerate.

- How to organize a loop
 - Common functions used in the for statement
 - range
 - Generate as many numbers as needed
 - Create and return an iterable object with a range of values equal to the given number.

Function	Description	Example
<code>range(stop)</code>	<ul style="list-style-type: none">• An integer from 0 to stop-1	<code>range(3)</code>
<code>range(start, stop)</code>	<ul style="list-style-type: none">• An integer from start to stop-1	<code>range(1, 4)</code>
<code>range(start, stop, step)</code>	<ul style="list-style-type: none">• an integer from start to stop-1, incremented by step	<code>range(2, 10, 2)</code>

Python - control statements

Types and uses of loops

Functions that are often used with for statements are range and enumerate.

- How to organize a loop
 - Common functions used in the for statement
 - range
 - Usage example

```
>>> add = 0
>>> for i in range(1, 11):
...     add = add + i
...
>>> print(add)
55
```

Python - control statements

Types and uses of loops

Functions that are often used with for statements are range and enumerate.

- How to organize a loop
 - Common functions used in the for statement
 - enumerate
 - Mean "to enumerate"
 - Pass the order and values of the list, if there is a list
 - Take an ordered data type (string, list, tuple, set, dictionary) as input and return an enumerate object containing the index values.

Python - control statements

Types and uses of loops

Functions that are often used with for statements are range and enumerate.

- How to organizing a loop
 - Common functions used in the for statement
 - enumerate
 - Usage examples

```
# Before using enumerate (which requires figuring out the list length and accessing the array by index)
```

```
data_list = ['a', 'b', 'c', 'd']
for i in range(len(data_list)):
    data = data_list[i]
    print('{}:{}'.format(i+1,data))
```

```
# After using enumerate
```

```
data_list = ['a', 'b', 'c', 'd']
for i , data in enumerate(data_list):
    print('{}:{}'.format(i+1,data))
```

Python - control statements

Types and uses of loops

List comprehension, which includes for statements within a list, can be used to create more convenient and intuitive programs. Python 2.7 has only list comprehension.

- How to organize a loop
 - List comprehension
 - Make programming easier and more intuitive by creating lists that embed for statements
 - Possible to omit the if conditional statement in a list-embedded syntax

Function	Description
1 for statement	[expression for item in iterable_object if conditional]
More than 1 for statement	[expression for item1 in iterable_object1 if conditional1 for item2 in iterable_object2 if conditional2 ... for itemn in iterable_objectn if conditionaln]

Python - control statements

Types and uses of loops

List comprehension, which includes for statements within a list, can be used to create more convenient and intuitive programs. Python 2.7 has only list comprehension.

- How to organize a loop
 - List comprehension example

```
# Before using list comprehension
```

```
a = [1,2,3,4]
result = []
for num in a:
    result.append(num*3)
print(result)
```

```
# After using list comprehension
```

```
a = [1,2,3,4]
result = [num * 3 for num in a]
print(result)
```

Python - functions

Python functions overview

A function is defined as a group of code that performs a specific function when used frequently in a program.

- What is a function?

- Reusable code
- Reduce program complexity
- Increase efficiency as programs become more modular

```
# Before using a function
sum = 0
for i in range(1,11):
    sum = sum + i
print(sum)

sum = 0
for i in range(11,21):
    sum = sum + i
print(sum)
```

```
# Define a function and create a program like this
def sum(x, y):
    result = 0
    for i in range(x,y+1):
        result = result + i
    return result

sum(1,10)
sum(11,20)

print(a)
print(b)
```

Python - functions

Python functions overview

A function is defined as a group of code that performs a specific function when used frequently in a program.

- Types of functions
 - User-defined functions
 - Functions arbitrarily defined by a user
 - Built-in functions
 - Predefined functions in a programming language
 - Possible to get help on a built-in function with `help([object])`
 - Object contains the function name.
 - Example

```
>>> help(len)
Help on built-in function len in module __builtin__:

len(...)
    len(object) -> integer

    Return the number of items of a sequence or collection.
```

Python - functions

Python function overview

`def` is a reserved word used to create the function, and the function name can be created by the user. The parameters in parentheses after the function name are variables that receive values passed as input to the function.

- Structure of Python functions

- Functions with no return value
 - Function declarations start with the reserved word `def` and end with a colon (`:`)
 - The function content is indented, and the function definition ends when the indentation ends.
 - The function name is arbitrarily assigned by the user
 - Parameters can have default values
 - If you don't specify an initial value, the function call must specify an argument
- Functions with return values
 - Include function content with no return value
 - Use the `return` command to return a value
 - When the `return` command is encountered, the function exits

```
def <function  
name>(parameter[=initial value],...):  
    statement to execute1  
    statement to execute2  
    ...
```

```
def <function  
name>(parameter[=initial value],...):  
    statements to execute1  
    statements to execute2  
    ...  
    return the return value
```

Python - functions

Using functions

It's important to remember that parameters and arguments are confusing terms that are used interchangeably. A parameter is a variable that receives the value passed as input to a function, while an argument is the input you pass when calling a function.

- Structure of Python functions

- Parameters and arguments
 - Parameters : variables that receive values passed as input to the function
 - Arguments : input values to pass when the function is called

```
>>> def Multi(a,b): # a, b are the parameters
... return<a*b>

>>> x = 2
>>> y = 10
>>> z = Multi(x,y) # x, y are the arguments
>>> print(z)
20
```

Python - functions

Python functions overview

Functions can be declared and used without return values.

- Structure of Python functions
 - Declare and use functions without return values
 - Functions with no input values

```
>>> def say():
...     print('Hi')

>>> say()
Hi
```

- Functions with input values

```
>>> def say(name):
...     print('My name is {}'.format(name))

>>> say("Hong Gil-dong")
My name is Gil-dong Hong.
```

Python - functions

Python functions overview

You can preset default values for parameters.

- Structure of Python functions
 - Declare and use functions without return values
 - Functions with preset default values for parameters

```
>>> def say(name, old=26):  
...     print('My name is {}'.format(name))  
...     print('I am {} years old.'.format(old))
```

```
>>> say("Hong Gil-dong")  
My name is Gil-dong Hong.  
I am 26 years old.
```

```
>>> say("Hong Gil-dong",30)  
My name is Hong Gil-dong.  
I am 30 years old.
```

Python - functions

Python functions overview

Functions can be declared and used with return values.

- Structure of Python functions
 - Declare and use functions without return values
 - Functions with input values

```
>>> def Multi(a,b):  
...     return a*b  
...  
>>> x = 2  
>>> y = 10  
>>> z = Multi(x,y)  
>>> print(z)  
20
```

Python - functions

Python functions overview

Functions can be declared and used with return values. Functions with multiple input values are written in the form of * parameters.

- Structure of Python functions

- Declare and use functions with return values
 - Functions with multiple input values
 - Parameters in parentheses changed to *parameters
 - A parameter name can be randomly chosen.

```
def function name (*parameters):  
    <statement to execute>  
    ...
```

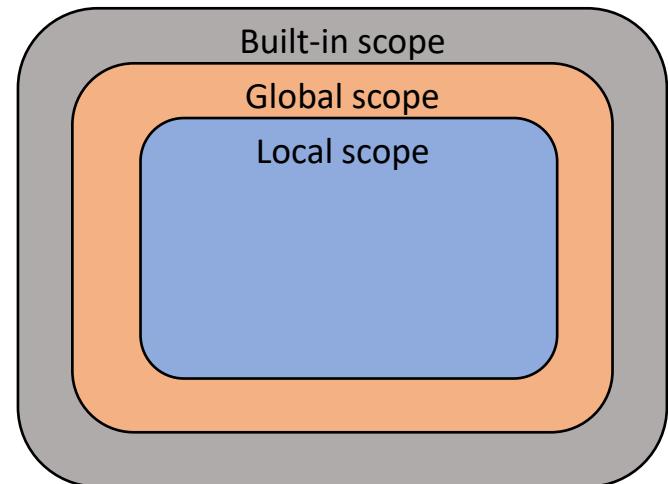
```
>>> def add_many(*args):  
...     result = 0  
...     for i in args:  
...         result = result + i  
...     return result  
  
>>> result = add_many(1,2,3)  
>>> print(result)  
6  
>>> result = add_many(1,2,3,4,5,6,7,8,9,10)  
>>> print(result)  
55
```

Python - functions

Using functions

Scoping rules are about the scope of variables declared within a function. Functions in Python have their own namespace. Python provides three types of namespaces : local, global, and built-in.

- Scoping rules
 - Rules about the scope of effect of variables declared inside functions
 - Python has separate namespaces
 - Namespace
 - The space where the variable name is stored
 - Organized into three types
 - Local scope : the place inside the function
 - Global scope : the place outside the function
 - Built-in scope : the place defined by Python itself



Python - functions

Using functions

The namespace search rule, also known as the LGB rule, is the local, global, and built-in order.

Scoping rules

- Namespace
 - Namespace search rules
 - Browse in the local scope → global scope → built-in scope order
 - Also known as the LGB rule, from the initials
 - Source code - "Scoping.py"

```
a = 100 # Variable a located outside the function (global scope)

def func1(x):
    return x + a # Find if the variable a exists in order L, G, B
print(func1(1))

def func2(x):
    a = 200 # Variable a located in the local scope of the function.
    return x + a # Find if the variable a exists in order L, G, B
print(func2(1))
```

<Result>
101
201

Python - functions

Using functions

Variables declared global change their scope from local to global.

- Scoping rules

- Namespace
 - Use the global command
 - Change the scope of the globally declared variable from local to global
 - Source code - "Scoping_global.py"

```
a = 100 # Variable a located in space outside the function (Global scope)

def func(x):
    global a # Declare global on variable a located in the local scope
    return x + a # Variable a in L changes its scope to G by declaring global

print(func(1))
```

<Result>
101

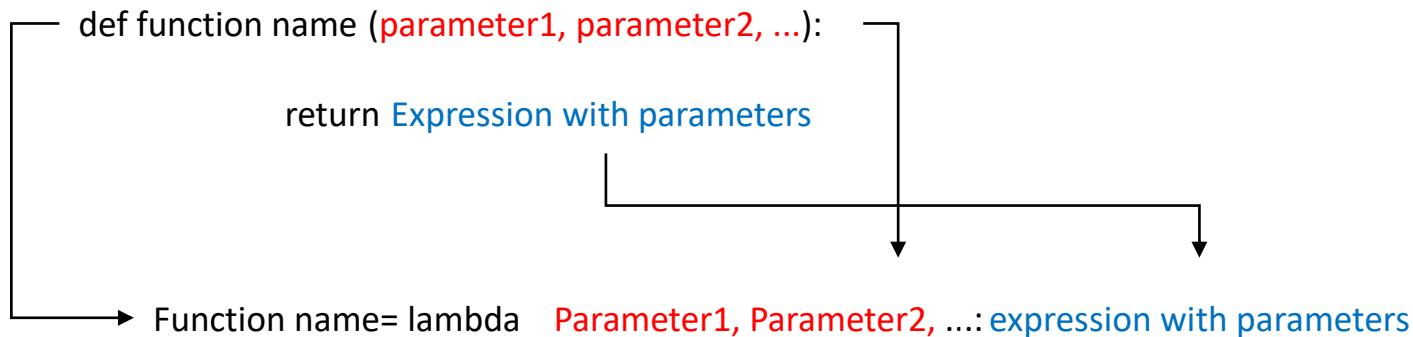
Python - functions

Using functions

Lambda is one of the reserved words in Python used to create functions, and plays the same role as def. It is usually used to keep functions single-line.

- Lambda

- One of the reserved words in Python that is used when creating functions.
- Assume the same role as def
- Usually used to condense a function into a single line.
- Used for things that are not complex enough to use def or where def is not available



Python - functions

Using functions

Lambda is one of the reserved words in Python used to create functions, and plays the same role as def. It is usually used to keep functions single-line.

- Lambda

- Usage examples

- Source file - "lambda.py"
 - Compare a function defined by the traditional def with a function defined by lambda (same result)

```
# A function defined by the existing def
def add(a, b):
    return a+b
result = add(3, 4)
print(result)
```

```
# A function defined by lambda
add = lambda a, b: a+b
result = add(3, 4)
print(result)
```

<Result>

7
7

Python - class

Classes and objects

We will learn and understand the terminology for classes.

- Class
 - A framework for creating objects
 - A collection of variables and functions
- Object
 - Data with specific properties and behaviors
 - Multiple objects can be created from a single class, but each must be unique from the other.
- Instance
 - An object created by a class
- The difference between objects and instances
 - In `a = cookie()`, `a` is an object and the object `a` is an instance of `cookie`.
 - An instance is a relational description of how a particular object (`a`) is an object of a particular class (`cookie`).

Python - class

Classes and objects

We will learn and understand the terminology for classes.

- Class variable
 - A variable that shares values across instances of all classes
- Instance variable
 - A variable that has different values for different instances
 - When used as a `self.variable`, it becomes an instance variable
- Method
 - A function-like concept
 - A function of a sort that is bound to a class and relates to instances within the class

Python - class

Classes and objects

Through the process of making one of South Korea's most popular street foods, a fish shaped bun (bungeo-ppang), we will come to understand the concept of class.

- Class and object examples

- Declare a class
 - Dough is necessary to make any kind of bungeo-ppang, so declare it as a class variable.
 - Class : bungeo-ppang mold
 - Class variable : dough

```
class bungeo-ppang mold:  
    dough = 10
```



Python - class

Classes and objects

Through the process of making one of South Korea's most popular street foods, a fish shaped bun (bungeo-ppang), we will come to understand the concept of class.

- Class and object examples

- Declare methods
 - Declare methods to determine the flavor of bungeo-ppang
 - Methods : red bean filling, chocolate filling
 - Instance variable : red bean

```
class bungeo-ppang mold:  
    dough = 10  
    def red bean filling(self,red bean):  
        self.red bean = red bean  
        result = bungeo-ppang mold.dough + self.red bean filling  
        print result  
  
    def chocolate filling(self,chocolate):  
        self.chocolate = chocolate  
        result = bungeo-ppang mold.dough + self.chocolate filling  
        print result
```



Python - class

Classes and objects

Through the process of making one of South Korea's most popular street foods, a fish shaped bun (bungeo-ppang), we will come to understand the concept of class.

- Class and object examples
 - Create an object (an instance of a class)
 - Create an object, i.e., an entity called bungeo-ppang
 - Combine methods on the created object to create bungeo-ppangs with different fillings
 - Object (an instance of a class) : bungeo-ppang

```
class bungeo-ppang mold:  
    dough = 10  
    def red bean filling(self,red bean):  
        self.red bean = red bean  
        result = bungeo-ppang mold.dough + self.red bean filling  
        print result  
  
    def chocolate filling(self,chocolate):  
        self.chocolate = chocolate  
        result = bungeo-ppang mold.dough + self.chocolate filling  
        print result  
  
bungeo-ppang = bungeo-ppang mold()  
bungeo-ppang.red bean filling(5)  
Bungeo-ppang.chocolate filling(5)
```



Python - class

Classes and objects

Through the process of making one of South Korea's most popular street foods, a fish shaped bun (bungeo-ppang), we will come to understand the concept of class.

- Class and object examples
 - Source code - "Class.py"

```
class case:  
    dough = 10  
    def make_red_bean(self, red_bean):  
        self.red_bean = red_bean  
        result = case.dough + self.red_bean  
        print(result)  
  
    def make_chocolate(self,chocolate):  
        self.chocolate = chocolate  
        result = case.dough + self.chocolate  
        print(result)  
bread = case()  
bread.make_red_bean(5)  
bread.make_chocolate(5)
```

Python - class

Create an arithmetic class

Let's practice using the class to create a calculator.

- How to create a class structure
 - Create a Calc class that contains only the sentence pass in the interactive interpreter.
 - In its current state, the Calc class does not contain any variables or functions, but it can create objects.
 - First create object a, and then check what type it is with type(a).
 - Object a is an object of the Calc class.

```
>>> class Calc:  
...     pass  
...  
>>> a = Calc()  
>>> type(a)  
<class '__main__.Calc'>
```

Python - class

Create an arithmetic class

You need to create a method to perform arithmetic functions.

- Create calculation capabilities on objects
 - The created object a has no functionality yet
 - Add arithmetic (+, -, *, /) functionality to the object a by creating a method
 - Create setdata method with three parameters (self, first, second) to calculate only two numbers

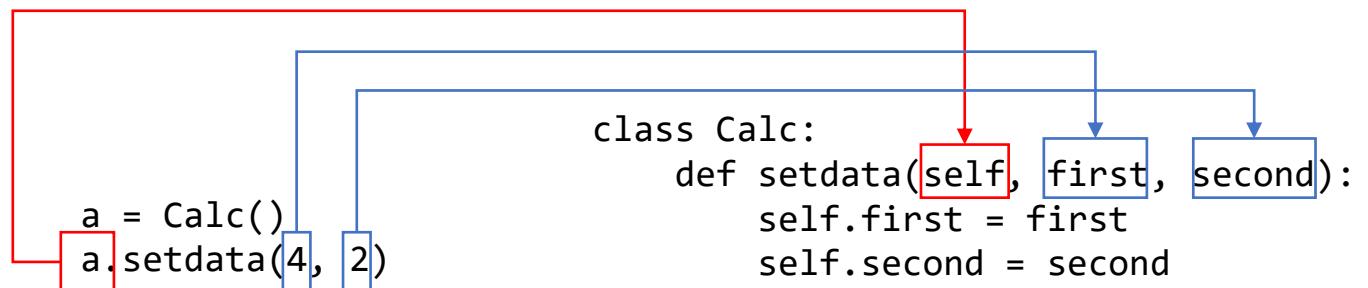
```
>>> class Calc:  
... def setdata(self, first, second):  
...     self.first = first  
...     self.second = second
```

Python - class

Create an arithmetic class

The setdata method takes three parameters : self, first, and second, but it actually passes only two values, such as a.setdata(4, 2), because the first parameter, self, is automatically passed by the object a that called the setdata method.

- How to create calculation capabilities on objects
 - The created object a has no functionality yet.
 - Create a setdata method with three parameters (self, first, second) to operate on only two numbers.
 - Create an object a and use it to call the setdata method through it as follows :
 - Pass the values 4 and 2 to the first and second parameters of the setdata method.



Python - class

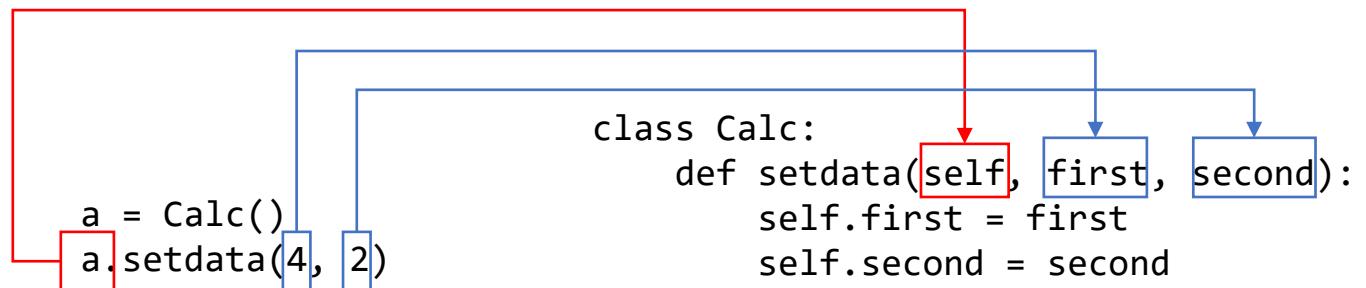
Create an arithmetic class

The setdata method takes three parameters : self, first, and second, but it actually passes only two values, such as a.setdata(4, 2), because the first parameter, self, is automatically passed by the object a that called the setdata method.

- How to create calculation capabilities on objects

- The meaning of self

- Explicitly implementing the first parameter of a method, self, is a unique feature of Python.
 - By convention, the first parameter name of a Python method is self.
 - You use self because when you call an object, you're passing the called object itself.
 - In fact, it doesn't matter if you use a name other than self.



Python - class

Create an arithmetic class

You add addition functionality to the setdata method.

- How to create calculation capabilities on objects
 - Create an arithmetic function.
 - add function (addition)

```
>>> class Calc:  
...     def setdata(self, first, second):  
...         self.first = first  
...         self.second = second  
...     def add(self):  
...         result = self.first + self.second  
...         return result  
...  
>>> a = Calc()  
>>> a.setdata(4, 2)  
>>> print(a.add())  
>>> 6
```

Python - class

Create an arithmetic class

You add subtraction functionality to the setdata method.

- How to create calculation capabilities on objects
 - Create an arithmetic function.
 - sub function (subtraction)

```
>>> class Calc:  
...     def setdata(self, first, second):  
...         self.first = first  
...         self.second = second  
...     def sub(self):  
...         result = self.first - self.second  
...         return result  
...  
>>> a = Calc()  
>>> a.setdata(4, 2)  
>>> print(a.sub())  
>>> 2
```

Python - class

Create an arithmetic class

You add multiplication functionality to the setdata method.

- How to create calculation capabilities on objects
 - Create an arithmetic function.
 - mul function (multiplication)

```
>>> class Calc:  
...     def setdata(self, first, second):  
...         self.first = first  
...         self.second = second  
...     def mul(self):  
...         result = self.first * self.second  
...         return result  
...  
>>> a = Calc()  
>>> a.setdata(4, 2)  
>>> print(a.mul())  
>>> 8
```

Python - class

Create an arithmetic class

You add division functionality to the setdata method.

- How to create calculation capabilities on objects
 - Create an arithmetic function.
 - div function (division)

```
>>> class Calc:  
... def setdata(self, first, second):  
...     self.first = first  
...     self.second = second  
... def div(self):  
...     result = self.first / self.second  
...     return result  
...  
>>> a = Calc()  
>>> a.setdata(4, 2)  
>>> print(a.div())  
>>> 2
```

Python - class

Create an arithmetic class

- How to create calculation capabilities on objects
 - Example of executing the add method without the setdata method on instance a of class Calc
 - The error "AttributeError: 'FourCal' object has no attribute 'first'" occurs.
 - The setdata method must be executed to create the first and the second object variables of the object a.

```
>>> a = Calc()
>>> a.add()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 6, in add
AttributeError: 'FourCal' object has no attribute 'first'
```

- When an object needs to have an initial value set, it's safer to implement an initializer than to call a method like setdata to set the initial value.

```
>>> a = Calc()
>>> a.setdata(4, 2)
>>> print(a.add())
>>> print(a.sub())
>>> print(a.mul())
>>> print(a.div())
```

Python - class

Initializer and constructor

When translating a constructor and an initializer, it is easy to distinguish between them in English words, but many reference books do not distinguish between initializers and constructors and use both as constructors.

- Initializer
 - If you create a constructor using the initializer(`__init__`) method, it is called an initializer.
- Constructor
 - if you use the constructor created by the initializer (`__init__`) method, it is called a constructor.
- How to write an initializer
 - Use `__init__` as the Python method name.
 - The `_` before and after `init` in the `__init__` method are two underscores (`_`).
 - The `__init__` method has the same properties as the `setdata` method except for the name.
 - However, because we named the method `__init__`, it is recognized as a constructor and is called automatically when the object is created.

Python - class

Initializer and constructor

- Initializer and constructor example
 - Source file - "Calc.py"

```
class Calc:  
    def __init__(self, first, second):  
        self.first = first  
        self.second = second  
    def setdata(self, first, second):  
        self.first = first  
        self.second = second  
    def add(self):  
        result = self.first + self.second  
        return result  
    def sub(self):  
        result = self.first - self.second  
        return result  
    def mul(self):  
        result = self.first * self.second  
        return result  
    def div(self):  
        result = self.first / self.second  
        return result  
a = Calc(4,2)  
print(a.add())  
print(a.sub())  
print(a.mul())  
print(a.div())
```

Python - class

Inheritance

Inheritance is categorized into parent class (super class), which inherits functionality, and child class (sub class), which is inherited by the functionality.

- Inheritance

- "To inherit," as in inheritance, when we say it to mean "to receive a fortune."
- Categorized into parent class (super class), which leaves functionality, and child class (sub class), which inherits the functionality.
- Child class can use the content of its parent class
- Writing format
 - Put parent class in parentheses when declaring a child class
 - Child class can have properties and methods of the parent class without listing them.

```
class ParentClass:  
    ...content...  
  
class ChildClass (ParentClass):  
    ...content...
```

Python - class

Inheritance

- Inheritance example
 - Source file - "Inheritance.py"
 - Child class that inherited properties and methods of parent class can use them.

```
class Country:  
    """Super Class"""\n\n    name = 'Country name'  
    population = 'Population'  
    capital = 'Capital'\n\n    def show(self):  
        print('This is a method of the country class.')\n\nclass Korea(Country):  
    """Sub Class"""\n\n    def __init__(self, name):  
        self.name = name\n\n    def show_name(self):  
        print('Country name is : ', self.name)\n\na = Korea('South Korea')  
a.show()  
a.show_name()
```

<Result>
Methods of the country class.
Country name is : South Korea

Python - class

Inheritance

Method overriding substitutes the existing method in the parent class with a new definition in the child class.

- Method overriding
 - Use to override a method of a parent class in a child class
 - Methods in the parent class are ignored and methods in the child class are executed.
 - Example : override the show method of a parent in the Korea class

```
class Korea(Country):
    """Sub Class"""

    def __init__(self, name, population, capital):
        self.name = name
        self.population = population
        self.capital = capital

    def show(self):
        print("Country: {}\nPopulation: {}\nCapital of the country: {}".format(self.name, self.population,
self.capital))

a = Korea('South Korea',50000000,'Seoul')
a.show()
```

Python - class

Inheritance

- Method overriding examples
 - Source file - "Methodoverriding.py"
 - Same as the parent class that exists in the Inheritance.py file
 - The show() method of the parent class is ignored and the show() method of the child class is executed.

```
...omitted
class Korea(Country):
    """Sub Class"""

    def __init__(self, name, population, capital):
        self.name = name
        self.population = population
        self.capital = capital

    def show(self):
        print("Country: {}\nPopulation: {}\nCapital of the
country: {}".format(self.name, self.population, self.capital))

a = Korea('South Korea', 50000000, 'Seoul')
a.show()
```

```
<Result>
Country: South Korea
Population: 50000000
Capital of a country: Seoul
```

Python - class

Inheritance

- Method overriding examples
 - Calling parent methods
 - Change the parent class of the existing Inheritance.py to "class Country(object):"
 - Call the parent class from within a child class using the super() keyword
 - Source file - "Methodoverriding2.py"

```
class Country(object):
...omitted

class Korea(Country):
    """Sub Class"""

    def __init__(self, name, population, capital):
        self.name = name
        self.population = population
        self.capital = capital

    def show(self):
        super().show()
        print("Country: {}\nPopulation: {}\nCapital of the
country: {}".format(self.name, self.population, self.capital))

a = Korea('South Korea',50000000,'Seoul')
a.show()
```

<Result>
Methods of the country class.
Country: South Korea
Population: 50000000
Capital of the country: Seoul

Python - modules and packages

Module

A module is a file that contains a collection of functions, variables, or classes. A module can also be thought of as a Python file that you make available for other Python programs to load and use.

- Module
 - Organize frequently used or useful code into logical chunks.
 - A file of Python code that contains a collection of functions, variables, or classes.
 - Typically, a Python .py file becomes a module.
 - Modules composed as files can be imported and used in different execution environments and different Python files.

Python - modules and packages

Module

import is a command that allows you to use a Python module you've already created.

- How to use a module

- Import a module to use it
- Import one or multiple modules
- import method
 - Be sure to enter the module name, such as 'module name.function name.'
 - Include everything in the module.
 - Use an as keyword to nickname a module.
- form import method
 - Module name can be omitted.
 - With *, everything in the module is included.
 - You can use as to nickname a function.

```
<import method>
import module
import module1, module2, module3 ...
import module name as nickname

<form import method>
from module import function
from module import function1, function2, function3 ...
from module import *
from module import function as nickname
```

Python - modules and packages

Module

Any Python file created with the .py extension is a module.

- How to create a module
 - Modules with add and sub functions
 - Source file - "module1.py"

```
def add(a, b):  
    return a + b  
  
def sub(a, b):  
    return a - b
```

Python - modules and packages

Module

- How to create a module
 - Use of modules
 - Bring up a module in the interactive interpreter and use the add and sub functions.

```
>>> import module1
>>> module1.add(3, 4)
7
>>> module1.sub(4, 2)
2
>>> from module1 import add
>>> add(3, 4)
7
>>> from module1 import sub
>>> sub(4, 2)
2
from module1 import add, sub
>>> add(3, 4)
7
>>> sub(4, 2)
2
from module1 import *
>>> add(3, 4)
7
>>> sub(4, 2)
2
```

Python - modules and packages

Module

You create a module with classes, functions, variables, and more.

- How to create a module
 - Modules with classes, functions, variables, and more
 - Source file - "module2.py"
 - Math class to calculate the area of a circle
 - add function to add two values
 - PI variable corresponding to the pi value

```
PI = 3.141592

class Math:
    def solv(self, r):
        return PI * (r ** 2)

def add(a, b):
    return a + b
```

Python - modules and packages

Module

- How to create a module
 - Use of a module
 - Load the module in the interactive interpreter and use the value of the PI variable in the module2.py file.

```
>>> import module2  
>>> module2.PI  
3.141592
```

- Use the Math class in moduel2.py.

```
>>> import module2  
>>> a = module2.Math()  
>>> a.solv(2)  
12.566368
```

- Use the add function in module2.py.

```
>>> import module2  
>>> module2.add(module2.PI,5)  
8.141592
```

Python - modules and packages

Module

- How to create a module
 - Find a module that contains if `_name_ == "__main__"`:
 - Source file - "module3.py"
 - Add code to print the results of `add(1, 4)` and `sub(4, 2)` in "module1.py"

```
def add(a, b):  
    return a+b  
  
def sub(a, b):  
    return a-b  
  
print(add(1, 4))  
print(sub(4, 2))
```

```
<Result>  
5  
2
```

- Problem occurs when importing "module3.py".
 - At the moment of importing module3, module3.py is executed and prints the result.

```
>>> import module3  
5  
2
```

Python - modules and packages

Module

If you execute the file directly, `__name__ == "__main__"` becomes True and the next statement in the if statement is executed. Conversely, if this module is called from an interactive interpreter or another file, `__name__ == "__main__"` becomes False and the next statement in the if statement is not executed.

- How to create a module
 - Find a module that contains if `__name__ == "__main__"`:
 - Source file - "module4.py"
 - How to resolve the issue
 - Use the `__name__` variable

```
def add(a, b):
    return a+b

def sub(a, b):
    return a-b

if __name__ == "__main__":
    print(add(1, 4))
    print(sub(4, 2))
```

Python - modules and packages

Module

If you execute the file directly, `__name__ == "__main__"` becomes True and the next statement in the if statement is executed. Conversely, if this module is called from an interactive interpreter or another file, `__name__ == "__main__"` becomes False and the next statement in the if statement is not executed.

- How to create a module
 - `__name__` variable
 - Special variable names used internally by Python
 - When you run the `module4.py` file directly, the `__name__` variable in `module4.py` stores the value of `__main__`.
 - When you import `module4` from the Python shell or another Python module, the `__name__` variable in `module4.py` stores the module name value `module4` from `module4.py`.

Python - modules and packages

Module

This is an example of importing a module from another file.

- How to import a module from other files
 - Source file - "module5.py"
 - Import module2 from module5.py and use the add function

```
import module2
result = module2.add(3, 4)
print(result)

...
# Same as above
from module2 import *
result = add(3, 4)
print(result)
...
```

Python - modules and packages

Module

This is an example of loading a module in a different location. We will move a random module for practice.

- How to import a module from a different location
 - Move module2.py to C:\acs_python\mymodule

```
C:\Users\ACS>cd C:\acs_python  
C:\acs_python>mkdir mymodule  
C:\acs_python>move .\code\module2.py mymodule  
1 file is moved.
```

Python - modules and packages

Module

This is an example of loading a module in a different location. Since we can only load files in the current directory or modules in the directory where the Python library is stored, we add the directory with `sys.path.append()`.

- How to import a module from a different location
 - Use `sys.path.append`
 - Use `sys.path` to determine the directory where the Python libraries are installed
 - Add the desired directory with `sys.path.append("directory")`

```
>>> import sys
>>> sys.path
[ '', 'C:\\\\Python27\\\\Lib\\\\idlelib', ...omitted..., 'C:\\\\Python27\\\\lib\\\\site-packages' ]
>>>
>>> sys.path.append("C:\\\\acs_python\\\\mymodule")
>>>
>>> sys.path
[ '', 'C:\\\\Python27\\\\Lib\\\\idlelib', ...omitted..., 'C:\\\\Python27\\\\lib\\\\site-packages',
'C:\\\\acs_python\\\\mymodule']
```

Python - modules and packages

Module

The string input and output functions `stdin.readline()` and `stdout.write()` in the `sys` module can be used to input and output strings.

- How to use a module
 - Input/output with the `sys` module
 - `stdin.readline()` function
 - Enter a string
 - `stdin.write()` function
 - Print the string

```
>>> import sys  
>>> input = sys.stdin.readline()  
'a'
```

- `stdin.write()` function
 - Print the string

```
>>> import sys  
>>> input = sys.stdin.readline()  
'a'  
>>> sys.stdout.write(input)  
'a'
```

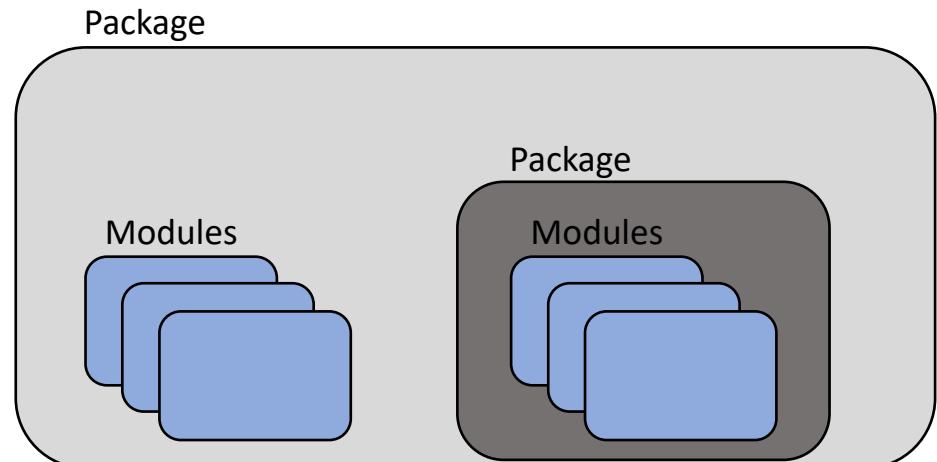
Python - modules and packages

Package

In Python, a module points to a single .py file, and a package is a collection of those modules, consisting of directories and modules.

- Package

- A set of modules placed in a single directory
- A directory usually contains a package initialization file called `_init_.py`.
- Manage Python modules in layers (directory structure) using dot (.)
- A package is a container for modules, which may contain other subpackages within it.



Python - modules and packages

Package

To build the package, you need to organize the directories and files as follows.

- How to create a package
 - Organize package folders
 - Configure a project directory under the C:\acs_python directory
 - Configure the main.py file and calcpkg directory in the project directory
 - Configure the __init__.py, operation.py, and geometry.py files in the calcpkg directory

```
project/
    main.py
    calcpkg/
        __init__.py
        operation.py
        geometry.py
```

Python - modules and packages

Package

To build the package, you need to organize the directories and files as follows.

- How to create a package
 - Organize modules in the calc pkg package
 - Operation module with arithmetic function
 - Source file - "operation.py"

```
def add(a, b):  
    return a + b  
  
def sub(a, b):  
    return a - b  
  
def mul(a, b):  
    return a * b  
  
def div(a, b):  
    return a / b
```

Python - modules and packages

Package

To build the package, you need to organize the directories and files as follows.

- How to create a package
 - Organize modules in the calc pkg package
 - Geometry module with the ability to calculate the area of a shape
 - Source file - "geometry.py"

```
def triangle_area(base, height):  
    return base * height / 2  
  
def rectangle_area(width, height):  
    return width * height
```

Python - modules and packages

Package

To build the package, you need to organize the directories and files as follows.

- How to use a package

- Configure the main.py module
 - Source file - "main.py"
 - How to import a package
 - import package.module
 - Package.module.variables
 - Package.module.functions()
 - Package.module.class()

```
import calcpkg.operation
import calcpkg.geometry

print(calcpkg.operation.add(20, 10))
print(calcpkg.operation.sub(20, 10))
print(calcpkg.operation.mul(20, 10))
print(calcpkg.operation.div(20, 10))

print(calcpkg.geometry.triangle_area(30, 40))
print(calcpkg.geometry.rectangle_area(30, 40))
```

```
<Result>
30
10
200
2
600
1200
```

Python - modules and packages

Package

To build the package, you need to organize the directories and files as follows.

- How to use a package

- Configure the main.py module
 - Source file - "main2.py"
 - How to import a package
 - from package.module import variable
 - from package.module import function
 - from package.module import class

```
from calc pkg.operation import add, mul
from calc pkg.geometry import *

print(add(20, 10))
print(mul(20, 10))

print(triangle_area(30, 40))
print(rectangle_area(30, 40))
```

```
<Result>
30
200
600
1200
```

Python - modules and packages

Package

Unconditional import, regardless of `__all__`, occurs when c, the last item in `from a.b.c import *`, is a module. In the case of `main3.py` below, we need to set `__all__` because `calcpkg` is a directory, not a module.

- Purpose of `__init__.py`
 - indicate that the directory is part of a package
 - If there is no `__init__.py` file in the directory containing the package, it will not be recognized as a package.
 - To import modules from a specific directory using `*`, set the `__all__` variable in the `__init__.py` file for that directory and specify which modules can be imported, as follows.

```
# C:\acs_python\project\calcpkg\__init__.py
__all__ = ['operation']
```

- Source code - "main3.py"

```
from calcpkg import *
print(operation.add(20, 10))
print(operation.mul(20, 10))
```

```
<Result>
30
200
```

Python - exception

Exception handling overview

An exception is an error that interrupts normal program flow, and exception handling is the mechanism for interrupting normal program flow and for continuing it in a surrounding context or block of code.

- Exception
 - An error that interrupts the normal program flow
- Exception Handling
 - A mechanism for interrupting normal program flow and continuing it in a surrounding context or block of code.
- Cases in which an exception is thrown (partial)

Operator	Description
SyntaxError	<ul style="list-style-type: none">• Typos, indentation mistakes
NameError	<ul style="list-style-type: none">• Access to undeclared variables
ZeroDivisionError	<ul style="list-style-type: none">• Division by '0'
IndexError	<ul style="list-style-type: none">• Exceeding of an accessible index in the list
TypeError	<ul style="list-style-type: none">• Unsupported operations

Python - exception

Exception handling overview

The major types of built-in exceptions are as follows.

- Major built-in exceptions

Operator	Description
Exception	<ul style="list-style-type: none">• Base class for all built-in exceptions, utilized when writing user-defined exceptions
ArithmeticError	<ul style="list-style-type: none">• Base class for numeric operation exceptions
LookupError	<ul style="list-style-type: none">• Base class for sequence-related exceptions
EnvironmentError	<ul style="list-style-type: none">• Basic class of Python external errors

Python - exception

Exception handling types

The except syntax can be used in three ways : a general except, an except with only the error that occurred, and an except with the error that occurred and an error message variable.

- try and except syntax
 - Basic structure for error handling
 - If an error occurs during the execution of the try block, an except block is performed
 - If no error occurs in the try block, the except block is not performed
 - Syntax used in except blocks
 - Common except statements
 - Except statements that include only errors that occurred
 - Except statements that include the error that occurred and the error message variable.

Python - exception

Exception handling types

- try and except syntax
 - Syntax used in except blocks
 - Common except statements
 - Perform an except block when an error occurs, regardless of the type of error

```
try:  
    ...  
except:  
    ...
```

- Source code - "except1.py"
 - Attempting to divide 9 by 0 throws a ZeroDivisionError, causing the except block to execute.
 - Can print a user-defined error message that says "An error occurred."

```
try:  
    9 / 0  
except:  
    print("An error occurred.")
```

<Result>
An error occurred.

Python - exception

Exception handling types

- try and except syntax
 - Syntax used in except blocks
 - Except statements that include only errors that occurred
 - When an error occurs, execute the except block only if the except statement matches a predetermined error name.

```
try:  
    ...  
except Error that occurred:  
    ...
```

- Source code - "except2.py"
 - Execute an except block only when a NameError occurs to output a user-specified error statement
 - The code below throws a ZeroDivisionError and cannot print the user-specified error text.

```
try:  
    9 / 0  
except NameError:  
    print("An error occurred.")
```

```
<Result>  
9 / 0  
ZeroDivisionError: integer division or modulo by  
zero
```

Python - exception

Exception handling types

- try and except syntax
 - Syntax used in except blocks
 - Except statements that include the error that occurred and the error message variable.
 - When an error occurs, execute the except block only if the except statement matches a predetermined error name.

```
try:  
    ...  
except Error that occurred as Error message variable:  
    ...
```

- Source code - "except3.py"
 - Attempting to divide 9 by 0 throws a ZeroDivisionError, causing the except block to execute.
 - Print an error message in variable e

```
try:  
    9 / 0  
except ZeroDivisionError as e:  
    print(e)
```

```
<Result>  
integer division or modulo by zero
```

Python - exception

Exception handling types

The else statement is executed if no exception is thrown, and the finally statement is executed whether an exception is thrown or not.

- else and finally examples

- The else statement is executed if no exception is thrown.
- The finally statement will be executed regardless of whether an exception is thrown or not.
- Source code - "else-finally.py"

```
def div(a,b):
    return a / b

try:
    c = div(5,2)
except ZeroDivisionError:
    print("The second argument must not be zero")
except TypeError:
    print("All arguments must be numbers")
except:
    print("ZeroDivisionError, any error other than TypeError")
else:
    print(c)
finally:
    print("Always execute a finally block")
```

```
<Result>
2
Always execute a finally block
```

Python - exception

Exception handling types

The pass command allows you to bypass the error, and the raise command allows you to raise the error.

- How to avoid an error
 - Use the pass command to pass when certain errors occur

```
try:  
    f = open("missingfile.txt", 'r')  
except FileNotFoundError:  
    pass
```

- How to raise an error
 - You can raise a specific error with the raise command.
 - Write the raise command in the form of the error ("error message") you want to raise

```
def convert(s):  
    try:  
        return int(s)  
    except (ValueError, TypeError) as e:  
        print(e)  
        raise ValueError("An invalid value was passed for the argument.")  
print(convert('s'))
```

Python - exception

Exception handling types

Exceptions are often created and used during the execution of a program to handle special cases in an exceptional way. The `_str_` method can be used to print an error message, just as `print(e)` creates a print statement.

- How to create an exception
 - Create an exception by inheriting from Python's built-in exception class
 - Implement the `_str_` method in the error class
 - Source code - "make_except.py"

```
class MyError(Exception):
    def __str__(self):
        return "An error occurred."

def animal(name):
    if name == 'Apple':
        raise MyError()
    print(name)

animal('Puppy')
animal('Apple')
```

```
<Result>
Puppy

Traceback (most recent call last):
  File "... omitted ...", line 12, in <module>
    animal('Apple')
  File "... Omitted ...", line 8, in animal
    raise MyError()
MyError: An error occurred.
```

Python - debugging

Debugging Python code with pdb

Python provides a Python debugger module called pdb for debugging. The debugger has many features, including step over, step into, breakpointing, callstack inspection, source listing, and variable substitution.

- Python debugger (pdb)
 - Python debugging tool
 - A tool to help you run the Python interpreter with line-by-line viewing
 - How to run pdb
 - Use it when you are running code from scratch

```
python -m pdb example.py
```

- Use it at a specific part of the code
 - After importing the pdb module, insert pdb.set_trace() where you want it to stop.

```
...  
import pdb;  
  
pdb.set_trace() # Insert at desired point  
...
```

Python - debugging

Debugging Python code with pdb

When you enter debugging mode using pdb, you'll see the (pdb) prompt, where you can use several pdb commands.

- How to use pdb

Operator	Description
<code>help</code>	<ul style="list-style-type: none">• Get help
<code>next</code>	<ul style="list-style-type: none">• Go to next line
<code>print</code>	<ul style="list-style-type: none">• Display variable values on the screen
<code>list</code>	<ul style="list-style-type: none">• Print source code list, mark the current position with an arrow
<code>where</code>	<ul style="list-style-type: none">• Print callstack
<code>continue</code>	<ul style="list-style-type: none">• Stop at the next breakpoint or run to the end without breakpoints
<code>step</code>	<ul style="list-style-type: none">• Getting inside a function
<code>return</code>	<ul style="list-style-type: none">• Run until just before the return of the current function
<code>!variable name = value</code>	<ul style="list-style-type: none">• Reset the variable value

Python - debugging

Debugging Python code with pdb

When you enter debugging mode using pdb, you'll see the (pdb) prompt, where you can use several pdb commands.

- Pdb usage examples
 - Source code - "debugging.py"

```
import pdb
def sum(x,y):
    return x + y

a = 10
pdb.set_trace()
b = 20
c = sum(a,b)
print(c)
```

Python - debugging

Debugging Python code with pdb

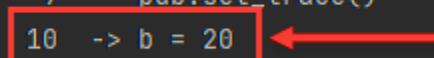
When you enter debugging mode using pdb, you'll see the (pdb) prompt, where you can use several pdb commands.

- Pdb usage examples

- Source code - "debugging.py"
 - list command : print a list of source code and mark your current position with an arrow

```
> c:\users\user\pycharmprojects\pythonproject\jbo\python_practice\print3.py(10)<module>()
-> b = 20
(Pdb) list
  5         return x + y
  6
  7
  8     a = 10
  9     pdb.set_trace()
10    -> b = 20
11     c = sum(a, b)
12     print(c)
[EOF]
(Pdb)
```

Current location



Python - debugging

Debugging Python code with pdb

When you enter debugging mode using pdb, you'll see the (pdb) prompt, where you can use several pdb commands.

- Pdb usage examples

- Source code - "debugging.py"
 - next command : go to the next line
 - step command : step inside a function

```
(Pdb) next
> c:\users\user\pycharmprojects\pythonproject\jbo\python_practice\print3.py(11)<module>()
-> c = sum(a, b)
(Pdb) step
--Call--
> c:\users\user\pycharmprojects\pythonproject\jbo\python_practice\print3.py(4)sum()
-> def sum(x, y):
(Pdb) next
> c:\users\user\pycharmprojects\pythonproject\jbo\python_practice\print3.py(5)sum()
-> return x + y
(Pdb) |
```

Python - debugging

Debugging Python code with pdb

When you enter debugging mode using pdb, you'll see the (pdb) prompt, where you can use several pdb commands.

- Pdb usage examples
 - Source code - "debugging.py"
 - where command : print callstack

```
(Pdb) where
  c:\users\user\pycharmprojects\pythonproject\jbo\python_practice\print3.py(11)<module>()
-> c = sum(a, b)
> c:\users\user\pycharmprojects\pythonproject\jbo\python_practice\print3.py(5)sum()
-> return x + y
(Pdb) |
```

Python - debugging

Debugging Python code with pdb

When you enter debugging mode using pdb, you'll see the (pdb) prompt, where you can use several pdb commands.

- Pdb usage examples

- return : execute until just before the return of the current function
- print : display variable values on screen
- !variable name=value : reset variable value

```
(Pdb) return
--Return--
> c:\users\user\pycharmprojects\pythonproject\jbo\python_practice\print3.py(5)sum()->30
-> return x + y
(Pdb) next
> c:\users\user\pycharmprojects\pythonproject\jbo\python_practice\print3.py(12)<module>()
-> print(c)
(Pdb) print(a,b,c)
10 20 30
(Pdb) !c = 300
(Pdb) next
300
--Return--
> c:\users\user\pycharmprojects\pythonproject\jbo\python_practice\print3.py(12)<module>()->None
-> print(c)
(Pdb) quit
```

04

Web programming

- Web programming overview
- Web programming environment setup
- HTML - introduction
- HTML - tags
- JavaScript - basics
- JavaScript - control statement
- JavaScript - functions
- JavaScript - implementing web page
- Understanding PHP language
- PHP - basic syntax
- PHP - DB connection
- PHP - implementing login page
- PHP – implementing board page

Web programming overview

Web programming overview

Web components include servers and clients that work as well as the following in real time while the user is on the Web.

- What is web programming?
 - The process of developing websites and web applications
 - This involves creating diverse features and interfaces that users can access via a web browser.
 - Goals : provide information, interact with users, manage data, etc.



Web programming overview

Frontend and backend

Web components include servers and clients that work as well as the following in real time while the user is on the Web.

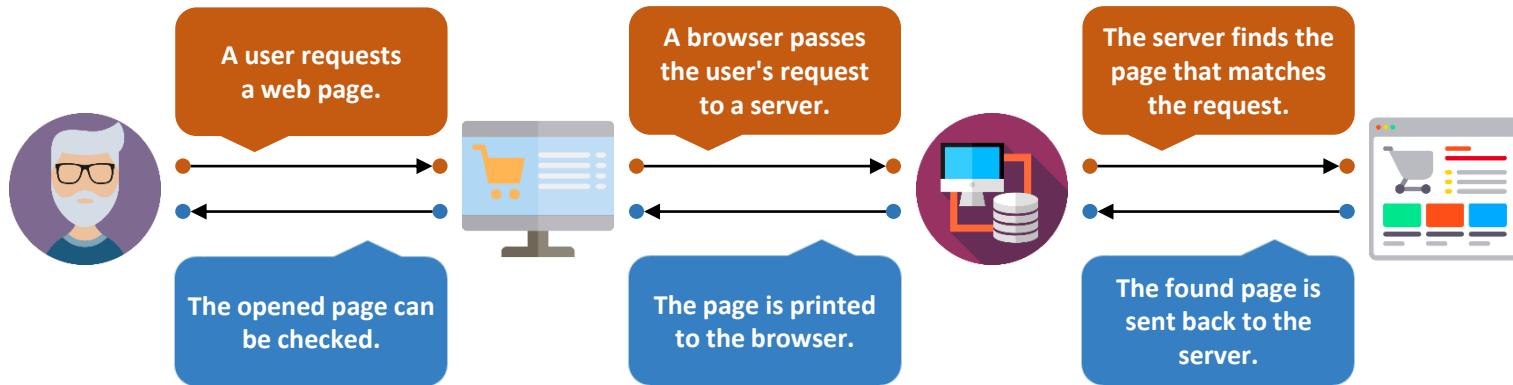
- Front-end development
 - Definition : develop the part of the web page that users directly interact with.
 - Techniques : HTML, CSS, JavaScript
 - Roles : implement User Interface (UI) and User Experience (UX) design, and responsive design
- Backend development
 - Definition : administer servers, databases, and server-side applications
 - Techniques : server-side languages (PHP, Python, Ruby, etc.) and database management systems (MySQL, MongoDB, etc.)
 - Roles : store/manage data, implement server logic, and develop APIs

Web programming overview

Client & server

Web components include servers and clients that work as well as the following in real time while the user is on the Web.

- Web client
 - Request a page from a web server, and print the server's response to the screen
- Web server
 - locate the requested page from a client, interpret/execute CGI, and send the results to the client in HTML, JavaScript, CSS, etc.

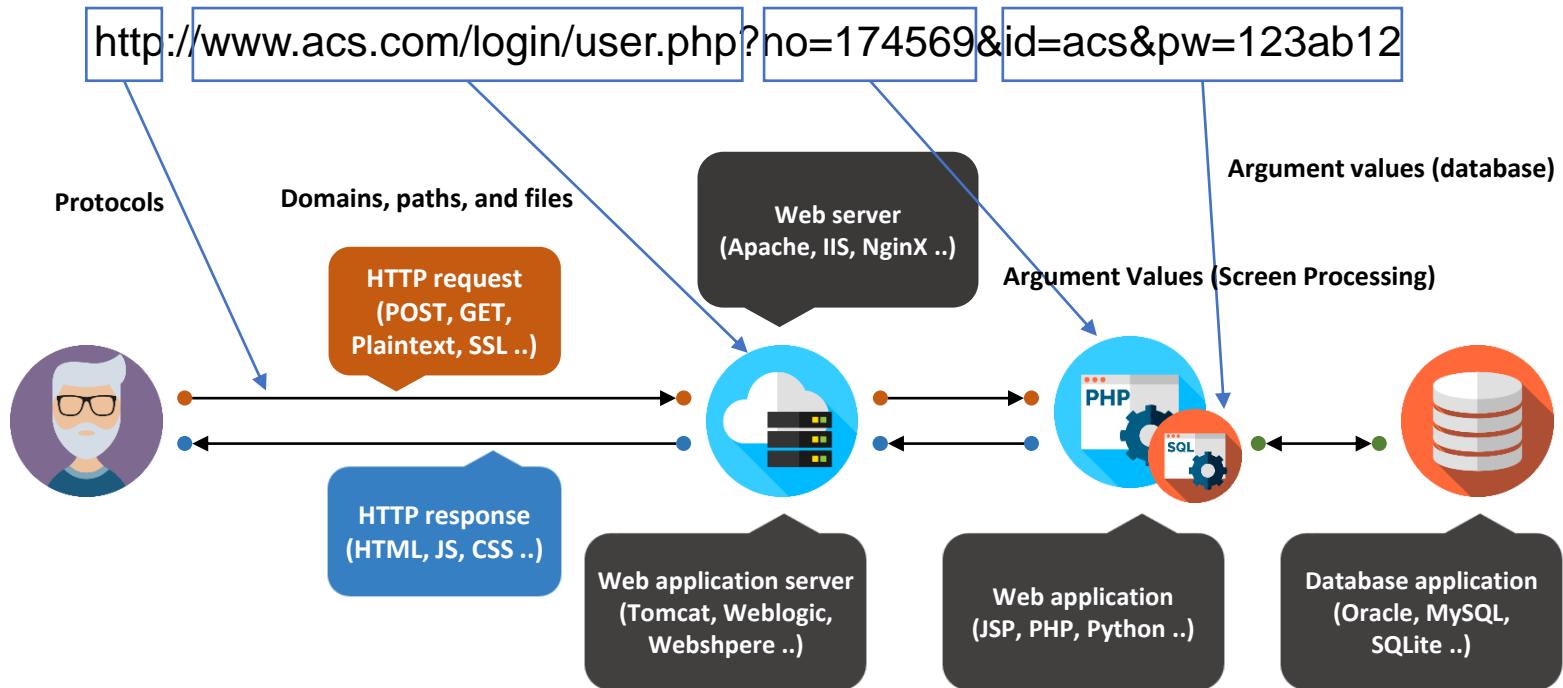


Web programming overview

Configuring web services

Web services process values in response to requests from the user's browser. Command processing is typically performed by a Web server that first receives the request, followed by a Web application implemented as a server-side script that processes the argument values, and then a database that stores the data required by the service.

- Understanding web components



Web programming overview

Configuring web services - languages

Client-side languages are primarily responsible for describing the structure or design directly visible to the user, or for sending requests to the server and presenting the retrieved data back to the user. Examples include HTML, JavaScript, and CSS.

- Client-side language
 - The language the user's browser handles
 - Features
 - Act as an interface between the user and the server
 - Provide interactive Web pages between the user and the server
 - Send requests to the server and retrieve data
 - Interact with ephemeral and local storage
 - Types
 - HyperText Markup (HTML) language : a language that is interpreted by browsers to create structure
 - JavaScript : a language that performs dynamic behaviors, such as computations, for HTML, which does not have these capabilities
 - Cascading Style Sheets (CSS) : also known as dependent sheets, used to present HTML beautifully

Web programming overview

Configuring web services - languages

The server-side language is not visible to the user, but it processes the user's requests and interacts with the database to process the data the user wants, which is then translated into the user-side language and sent to the user.

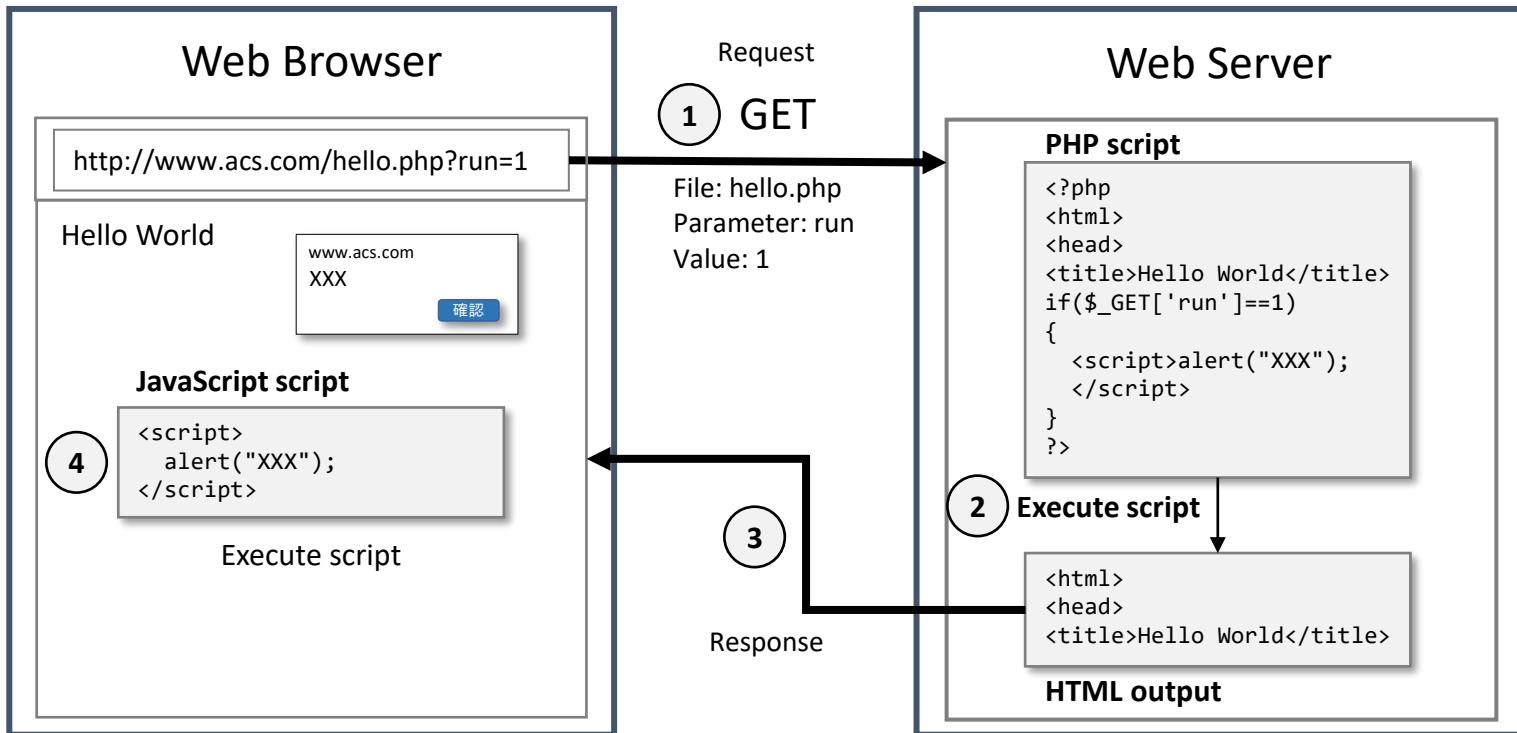
- Server-side language
 - Languages that work on the server
 - Features
 - Process user input (parameters)
 - Deliver the requested page
 - Interacting with the database
 - Convert processed data to user-side language
 - The computation is invisible to the user
 - Types
 - PHP, ASP.NET (C# or Visual Basic), C++, Java and JSP, Python, Ruby on Rails and so on.

Web programming overview

Configuring web services - languages

In a web service, client-side and server-side languages work together to process the data the user wants and present it in the user's browser.

- The linguistic flow of the web service composition



Web programming environment setup

Lab exercise for building a web server (Apache + PHP + MariaDB)

For the lab exercise for building web server, we will use the Apache web server and MariaDB on top of CentOS Linux. The language will be PHP.

- Apache web server
 - HTTP web server managed by the Apache Software Foundation
 - Able to run on a variety of OSes including BSD, Linux, Unix, and Windows
 - Highly scalable
 - The module concept allows you to add tons of functionality and integrate with other programs.
- MariaDB
 - Open source Relational DataBase Management System (RDBMS)
 - It is based on the same source code as MySQL and uses exact command matching to maintain high compatibility with MySQL.

Web programming environment setup

Lab exercise for building a web server (Apache + PHP + MariaDB)

- Web server deployment environments and installation version (based on lab content)
 - OS : CentOS Linux release 7.6.1810 (Core)
 - Apache : Apache/2.4.6 (CentOS)
 - PHP : PHP 5.4.16
 - MariaDB : mysql Ver 15.1 Distrib 5.5.64-MariaDB, for Linux (x86_64)
- Install APM yum automatically
 - If you run an automated installation with yum, it will install the necessary dependency libraries for the installation.
 - Use the -y option to the yum command to answer "yes" to all of the prompts during the installation.

```
$ sudo yum install -y httpd php mariadb-server php-mysql
Loaded plugins: fastestmirror, refresh-packagekit, security
Setting up Install Process
Loading mirror speeds from cached hostfile
 * base: data.aonenetworks.kr
 * extras: data.aonenetworks.kr
 * updates: data.aonenetworks.kr
:
```

Web programming environment setup

Lab exercise for building a web server (Apache + PHP + MariaDB)

- Set up autorun on service boot
 - Set up Apache autorun
 - httpd on CentOS and apache2 on Ubuntu

```
$ sudo systemctl enable httpd
Created symlink from /etc/systemd/system/multi-user.target.wants/httpd.service to ...
```

- Set up MariaDB autorun

```
$ sudo systemctl enable mariadb
Created symlink from /etc/systemd/system/multi-user.target.wants/mariadb.service to ...
```

- Start a service
 - Start the Apache service

```
$ sudo service httpd start or sudo systemctl start httpd
Redirecting to /bin/systemctl start httpd.service
```

- Start the MariaDB service
- ```
$ sudo service mariadb start or sudo systemctl start mariadb
Redirecting to /bin/systemctl stop mariadb.service
```

# Web programming environment setup

Lab exercise for building a web server (Apache + PHP + MariaDB)

## ● Initial setup of the MariaDB service

```
$ sudo mysql_secure_installation
:
Enter current password for root (enter for none): Enter # Without a root password for MariaDB for now, enter
OK, successfully used password, moving on...
:
Set root password? [Y/n] y # Set MariaDB root password query
New password: acs123123 # Enter the root password to set
Re-enter new password: acs123123 # Confirm the root password you set and re-enter it again
Password updated successfully!
Reloading privilege tables..
... Success!
:
Remove anonymous users? [Y/n] y # Query for anonymous access, will block it for security
... Success!
:
Disallow root login remotely? [Y/n] y # Query for root's remote access control, will block it for security.
... Success!
:
Remove test database and access to it? [Y/n] y # Query whether to delete the database created for testing
- Dropping test database...
... Success!
- Removing privileges on test database...
... Success!
:
Reload privilege tables now? [Y/n] y # Query for applicability to currently set values
... Success!
```

# Web programming environment setup

Lab exercise for building a web server (Apache + PHP + MariaDB)

- Add a character format setting in the MariaDB configuration file

```
$ sudo vi /etc/my.cnf
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
symbolic-links=0
init_connect="SET collation_connection = utf8_general_ci"
init_connect="SET NAMES utf8"
character-set-server = utf8
collation-server = utf8_general_ci

[client]
port=3306
default-character-set = utf8

[mysqldump]
default-character-set = utf8

[mysql]
default-character-set = utf8

[mysqld_safe]
log-error=/var/log/mariadb/mariadb.log
pid-file=/var/run/mariadb/mariadb.pid

!includedir /etc/my.cnf.d
```

# Web programming environment setup

Lab exercise for building a web server (Apache + PHP + MariaDB)

- Check MariaDB Settings

```
$ mysql -u root -p
Enter password: acs123123 # Enter the root password you set when running mysql_secure_installation
Welcome to the MariaDB monitor. Commands end with ; or \g.
:
MariaDB [(none)]> show variables like 'c%'; # Check the current character format settings in MariaDB
+-----+-----+
| Variable_name | Value |
+-----+-----+
| character_set_client | utf8
| character_set_connection | utf8
| character_set_database | utf8
| character_set_filesystem | binary
| character_set_results | utf8
| character_set_server | utf8
| character_set_system | utf8
| character_sets_dir | /usr/share/mysql/charsets/
| collation_connection | utf8_general_ci
| collation_database | utf8_general_ci
| collation_server | utf8_general_ci
| completion_type | NO_CHAIN
| concurrent_insert | AUTO
| connect_timeout | 10
+-----+-----+
14 rows in set (0.00 sec)

MariaDB [(none)]>
```

# Web programming environment setup

Lab exercise for building a web server (Apache + PHP + MariaDB)

- CentOS firewalld, iptables settings
  - Default use of firewalld instead of iptables as firewall starting with CentOS 7
  - Exit iptables and set up firewalld
    - check the iptables service

```
$ sudo service iptables status or sudo systemctl status iptables
```

- If the iptables service is running, set it to stop running automatically on boot after shutdown

```
$ sudo service iptables stop or sudo systemctl stop iptables
$ sudo systemctl disable iptables
```

- Check the firewalld service

```
$ sudo service firewalld status or sudo systemctl status firewalld
```

- Start the firewalld service if it is stopped and set it to run automatically at boot time

```
$ sudo service firewalld start or sudo systemctl start firewalld
$ sudo systemctl enable firewalld
```

# Web programming environment setup

Lab exercise for building a web server (Apache + PHP + MariaDB)

- CentOS firewalld, iptables settings
  - Exit iptables and set up firewalld
    - If you are running the firewalld service, add the http service or port 80.

```
$ sudo firewall-cmd --permanent --zone=public --add-service=http # Permanently add the http service to the default zone, public
$ sudo firewall-cmd --permanent --zone=public --add-port=80/tcp # Permanently add port 80/tcp to the default zone, public
$ sudo firewall-cmd --reload # Restart firewalld
$ sudo firewall-cmd --list-all # Check the list of firewalld settings
```

- Check the operation of Apache + PHP
  - Create a test page file

```
$ sudo vi /var/www/html/index.php # Here is the Apache default DocumentRoot and where web page files exist: /var/www/html
<?php phpinfo() ?>
$ ls -al /var/www/html/index.php
-rw-r--r-- 1 root root 986 Feb 11 13:10 /var/www/html/index.php
```

# Web programming environment setup

Lab exercise for building a web server (Apache + PHP + MariaDB)

- Check the operation of Apache + PHP
  - Check the web page access
    - Check the IP set in the OS and see the operation via a web browser on the host PC

The screenshot shows a web browser window displaying the output of the PHPinfo() function. The title bar reads "phpinfo()". The main content area has a header "PHP Version 5.4.16" with a "php" logo. Below this is a table with numerous rows of configuration parameters. Some key entries include:

| System                                  | Linux localhost.localdomain 3.10.0-957.el7.x86_64 #1 SMP Thu Nov 8 23:39:32 UTC 2018 x86_64                                                                                                                                                                                                          |
|-----------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Build Date                              | Nov 1 2019 16:05:03                                                                                                                                                                                                                                                                                  |
| Server API                              | Apache 2.0 Handler                                                                                                                                                                                                                                                                                   |
| Virtual Directory Support               | disabled                                                                                                                                                                                                                                                                                             |
| Configuration File (php.ini) Path       | /etc                                                                                                                                                                                                                                                                                                 |
| Loaded Configuration File               | /etc/php.ini                                                                                                                                                                                                                                                                                         |
| Scan this dir for additional .ini files | /etc/php.d                                                                                                                                                                                                                                                                                           |
| Additional .ini files parsed            | /etc/php.d/curl.ini, /etc/php.d/fileinfo.ini, /etc/php.d/gd.ini, /etc/php.d/json.ini, /etc/php.d/mbstring.ini, /etc/php.d/mysql.ini, /etc/php.d/mysqli.ini, /etc/php.d/pdo.ini, /etc/php.d/pdo_mysql.ini, /etc/php.d/pdo_sqlite.ini, /etc/php.d/phar.ini, /etc/php.d/sqlite3.ini, /etc/php.d/zip.ini |
| PHP API                                 | 20100412                                                                                                                                                                                                                                                                                             |
| PHP Extension                           | 20100525                                                                                                                                                                                                                                                                                             |
| Zend Extension                          | 220100525                                                                                                                                                                                                                                                                                            |
| Zend Extension Build                    | API220100525.NTS                                                                                                                                                                                                                                                                                     |
| PHP Extension Build                     | API20100525.NTS                                                                                                                                                                                                                                                                                      |
| Debug Build                             | no                                                                                                                                                                                                                                                                                                   |
| Thread Safety                           | disabled                                                                                                                                                                                                                                                                                             |
| Zend Signal Handling                    | disabled                                                                                                                                                                                                                                                                                             |
| Zend Memory Manager                     | enabled                                                                                                                                                                                                                                                                                              |
| Zend Multibyte Support                  | provided by mbstring                                                                                                                                                                                                                                                                                 |

# HTML - introduction

## Understanding HTML with a lab exercise

HyperText Markup Language (HTML) is a markup language, not a programming language, that defines conventions for describing web pages to indicate the format in which a document is displayed on a screen or to specify the logical structure of data.

- HTML overview
  - What HyperText Markup Language (HTML) means
    - HyperText : text that beyond being simple contains concepts such as links
    - Markup language
      - Specify the format in which a document appears on the screen or the logical structure of data
      - HTML uses tags with angle brackets (<, >), such as "<body>Hello World</body>".
  - HTML tag format

```
<Tag Option1="Value1" Option2="Value2" >String</Tag>
```

- Tag names are enclosed in angle brackets (>, <), with opening and closing tags.
  - The opening tag is simply an angle bracket, consisting of the tag name only.
  - The closing tag is an angle bracket, a forward slash (/) before the tag name, and the tag name.
- A space, even multiple spaces in a row, in a string inside a tag is treated as a single space.

# HTML - introduction

## Understanding HTML with a lab exercise

- HTML basic format

```
<!DOCTYPE HTML>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Hello HTML</title>
 </head>
 <body>
 <p>Hello World!</p>
 </body>
</html>
```

- <!DOCTYPE HTML> : declaration tag in a HTML5 document type
- <html> ~ </html> : indicate the beginning and end of a web page
- <head> ~ </head>
  - Specify attributes of HTML documents (used in related tags such as <meta>, <title>, <link>)
    - <title> ~ </title> : document title tag
    - <meta> : tag for character encoding, document keywords, and summary information
- <body> ~ </body>
  - Contain the actual content displayed in a browser, and most of tags are used below these tags.

# HTML - tags

## Understanding HTML with a lab exercise

- HTML key tags
  - Text-related tags

Tag	Description
<h1 ~ h6> String </h1 ~ h6>	<ul style="list-style-type: none"><li>• The tags that express the header as it appears in the browser</li><li>• The &lt;h1&gt; tag is the largest, and the &lt;h6&gt; tag is the smallest tags.</li></ul>
<p> String </p>	<ul style="list-style-type: none"><li>• Paragraph break tags</li></ul>
<b> String </b>	<ul style="list-style-type: none"><li>• Tags to thicken text (bold)</li></ul>
<i> String </i>	<ul style="list-style-type: none"><li>• Tags to italicize text</li></ul>
 	<ul style="list-style-type: none"><li>• Line break tag</li></ul>
<hr>	<ul style="list-style-type: none"><li>• Horizontal line generation tag</li></ul>

- List-related tags

Tag	Description
<ul> ~ </ul>	<ul style="list-style-type: none"><li>• Tags to display an unordered list</li></ul>
<ol> ~ </ol>	<ul style="list-style-type: none"><li>• Tags to display an ordered list</li></ul>
<li> String </li>	<ul style="list-style-type: none"><li>• Tags to mark each item in a list</li></ul>

# HTML - tags

## Understanding HTML with a lab exercise

- HTML key tags
  - Lab exercise using text and list-related tags

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>HTML Tag Example</title>
 </head>
 <body>
 <p>
 <h1>This is heading 1</h1>
 <h2>This is heading 2</h2>
 </p>
 <p>

 This is the first list in unordered list.
 <i>This is the second list in unordered list.</i>

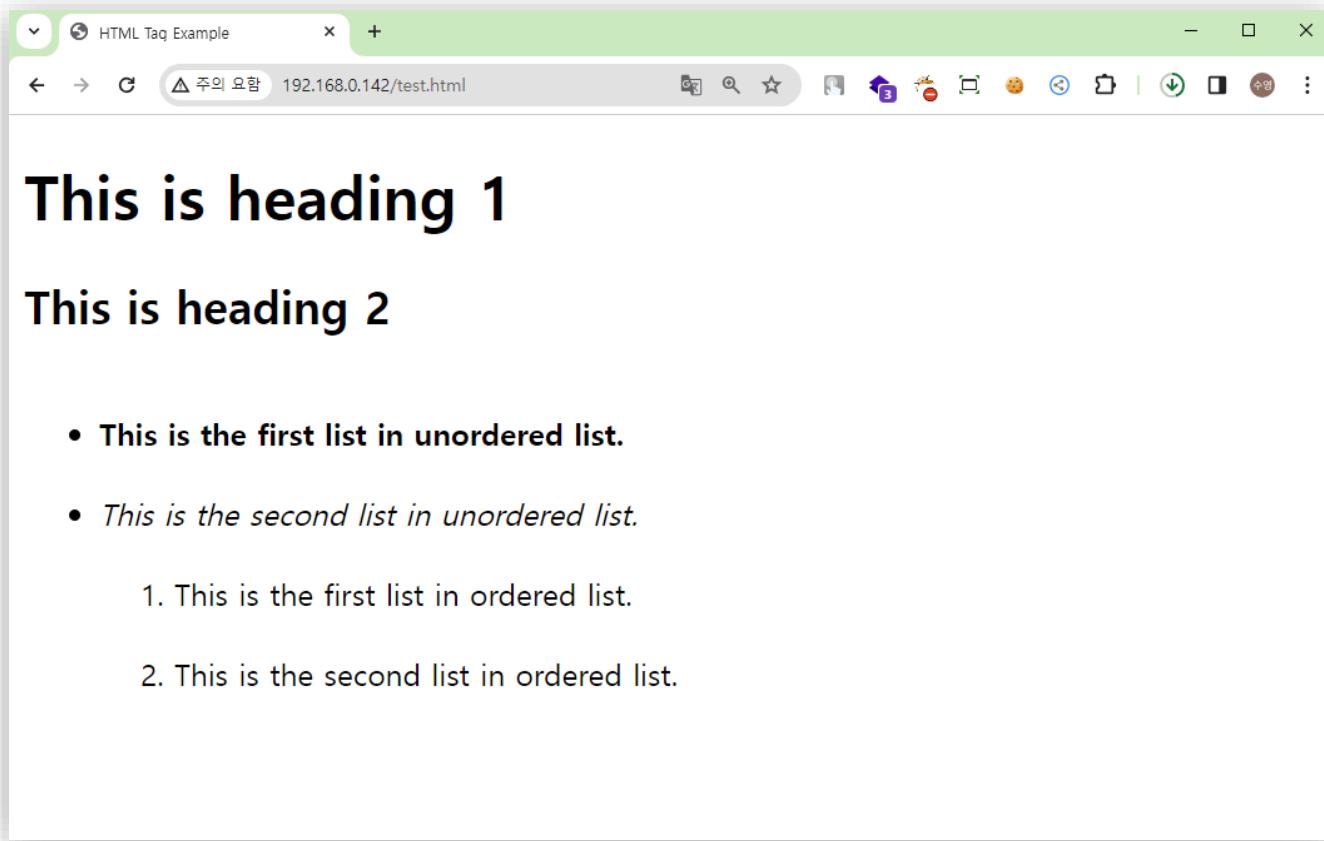
 This is the first list in ordered list.
 This is the second list in ordered list.

 </p>
 </body>
</html>
```

# HTML - tags

## Understanding HTML with a lab exercise

- HTML key tags
  - lab exercise using text and list-related tags
    - Save the file you created and check it by accessing <http://my ip/test.html> in a browser.



# HTML - tags

## Understanding HTML with a lab exercise

- HTML key tags
  - Tags for links and images

Tag	Description
<code>&lt;a href="Value"&gt; String &lt;/a&gt;</code>	<ul style="list-style-type: none"><li>• Tags that create hyperlinks</li></ul>
<code>&lt;img src="Value" alt="Value" title="Value" height="Value" width="Value"&gt;</code>	<ul style="list-style-type: none"><li>• Tags that add images to the page</li><li>• img tag options<ul style="list-style-type: none"><li>▪ src : specify the path to the image file<ul style="list-style-type: none"><li>- Specify an absolute path URL : http:// or https://</li><li>- download/image/file : Specify a relative path</li><li>- data:image/jpeg png gif;base64, A1bC2dE3fG... : specify the file in base64 encoded form</li></ul></li><li>▪ alt : give a description of the image if you can't see it</li><li>▪ title : give additional information about the image<ul style="list-style-type: none"><li>- Speech bubble on a mouse hover over an image</li></ul></li><li>▪ height : specify the vertical width of the image (in pixels)</li><li>▪ width : specify the horizontal width of the image (in pixels)</li></ul></li></ul>

# HTML - tags

## Understanding HTML with a lab exercise

- HTML key tags
  - Lab exercise on tagging for links and images

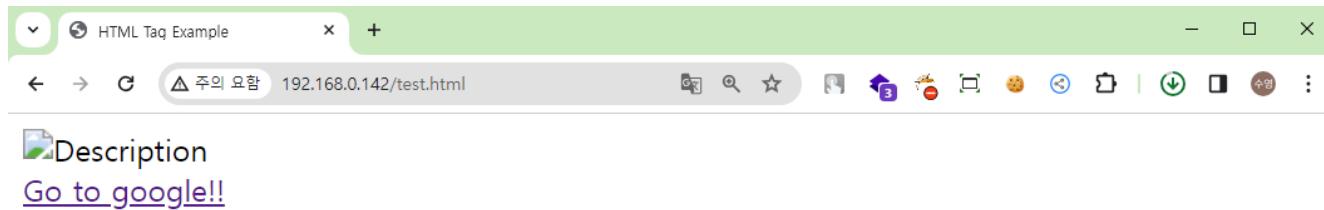
```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>HTML Tag Example</title>
 </head>
 <body>

 Go to google!!
 </body>
</html>
```

# HTML - tags

Understanding HTML with a lab exercise

- HTML key tags
  - Lab exercise with link image related tags
    - Save the file you created and check it by visiting <http://my ip/test.html> in a browser.



# HTML - tags

## Understanding HTML with a lab exercise

- HTML key tags
  - Table-related tags

Tag	Description
<table> ~ </table>	<ul style="list-style-type: none"><li>• Tags to create a table</li></ul>
<thead> ~ </thead> <tbody> ~</tbody> <tfoot> ~ </tfoot>	<ul style="list-style-type: none"><li>• Table's detailed structure such as top, body, and bottom tags</li></ul>
<caption> String </caption>	<ul style="list-style-type: none"><li>• Title tags in a table</li></ul>
<tr> ~ </tr>	<ul style="list-style-type: none"><li>• Row tags in a table</li></ul>
<th> String </th>	<ul style="list-style-type: none"><li>• Header tags for rows and columns in a table</li><li>• Center and bold by default</li></ul>
<td> String </td>	<ul style="list-style-type: none"><li>• Expression of the table's cells, the content of a table</li></ul>

# HTML - tags

## Understanding HTML with a lab exercise

- HTML key tags
  - Table-related tag properties

Tag	Description
<code>&lt;table&gt;, &lt;tr&gt;, &lt;td&gt; &lt;thead&gt;, &lt;tbody&gt;, &lt;tfoot&gt;</code>	<ul style="list-style-type: none"><li>• Alignment : align="alignment method" (left /right/center, etc.)</li><li>• Specify width : width="number" (width, in px)<ul style="list-style-type: none"><li>- &lt;thead&gt;, &lt;tbody&gt;, and &lt;tfoot&gt; tags cannot be specified.</li></ul></li><li>• Specify height : height="number" (vertical length, in px)<ul style="list-style-type: none"><li>- &lt;thead&gt;, &lt;tbody&gt;, and &lt;tfoot&gt; tags cannot be specified.</li></ul></li><li>• Specify background image : background="filepath"</li><li>• Background coloring : bgcolor="color"</li></ul>
<code>&lt;table&gt;</code>	<ul style="list-style-type: none"><li>• Specify a table border thickness : border="number"</li><li>• Specify padding inside cells : cellpadding="number"</li><li>• Specify the spacing between cells : cellspacing="number"</li></ul>
<code>&lt;td&gt;, &lt;th&gt;</code>	<ul style="list-style-type: none"><li>• Merge row columns by number (vertically) : rowspan="number"</li><li>• Merge columns by number (horizontally) : colspan="number"</li></ul>

# HTML - tags

## Understanding HTML with a lab exercise

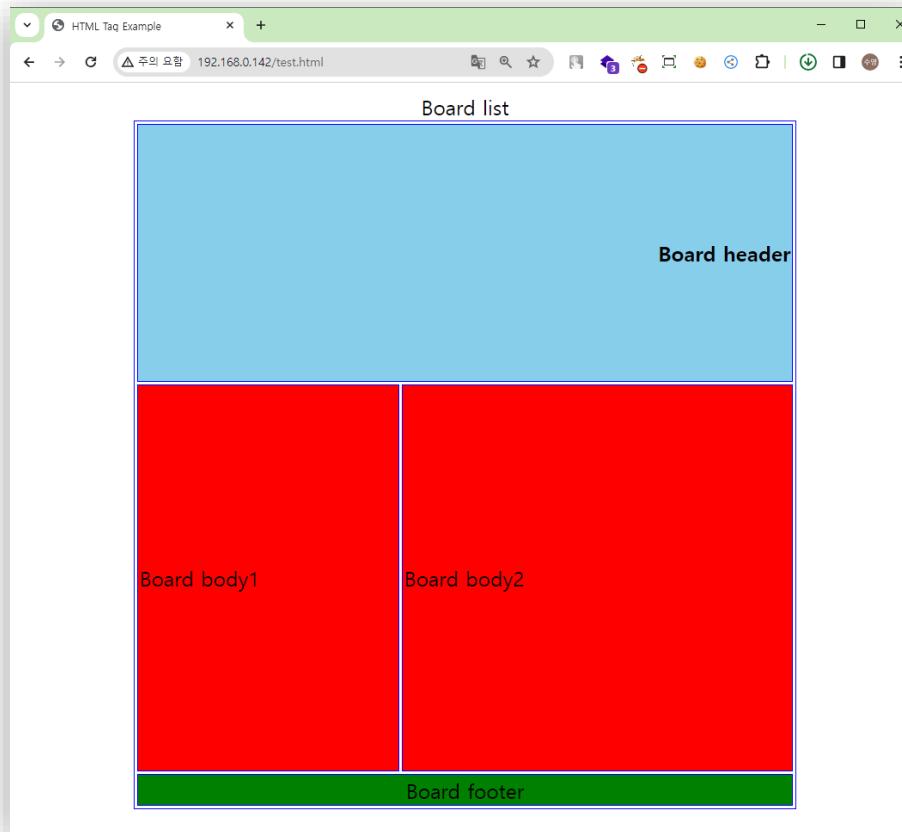
- HTML key tags
  - Lab exercise with table-related tags

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>HTML Tag Example</title>
 </head>
 <body>
 <table border="1" bordercolor="blue" align="center">
 <caption>Board list</caption>
 <thead bgcolor="skyblue" align="right">
 <tr height="200"> <th colspan="4">Board header</th> </tr>
 </thead>
 <tbody bgcolor="red">
 <tr height="300">
 <td colspan="2" width="200">Board body1</td>
 <td colspan="2" width="300">Board body2</td>
 </tr>
 </tbody>
 <tfoot bgcolor="green" align="center">
 <tr> <td colspan="4">Board footer</td> </tr>
 </tfoot>
 </table>
 </body>
</html>
```

# HTML - tags

Understanding HTML with a lab exercise

- HTML key tags
  - Lab exercise with table-related tags
    - Save the file you created and check it by visiting <http://my ip/test.html> in a browser.



# HTML - tags

## Understanding HTML with a lab exercise

- HTML key tags
  - Form-related tags
    - Take input to interact with ASP, PHP, JSP, etc.

Tag	Description
<form> ~ </form>	<ul style="list-style-type: none"><li>• Specify an area to receive input values</li><li>• You can have multiple &lt;form&gt; on a page, but not areas within areas.</li></ul>
<input type="Value">	<ul style="list-style-type: none"><li>• Specify input value elements</li><li>• Specify different input values based on type</li></ul>
<textarea> String </textarea>	<ul style="list-style-type: none"><li>• Specify multi-line string values</li></ul>

- <form> tag key properties

Property	Description
action="Value"	<ul style="list-style-type: none"><li>• Specify a server-side file to send form input values to</li></ul>
name="Value"	<ul style="list-style-type: none"><li>• Give your form a name to identify it</li></ul>
method="GET or POST"	<ul style="list-style-type: none"><li>• Specify the http method to send the form to the server (GET or POST)</li></ul>

# HTML - tags

## Understanding HTML with a lab exercise

- HTML key tags
  - Form-related tags
    - <input> tag key properties

Property	Description
type="Value" (required attribute)	<ul style="list-style-type: none"><li>• Specify in what format you want to receive information</li><li>• Input tags with the same type attribute can be distinguished from each other by specifying them as the id attribute.</li></ul>
placeholder="Value"	<ul style="list-style-type: none"><li>• Explained by default</li><li>• If the user types it, the entry is deleted.</li></ul>
readonly="Value"	<ul style="list-style-type: none"><li>• Make fields readable only</li></ul>
required	<ul style="list-style-type: none"><li>• Specify required input fields</li></ul>
max(min)length="Value"	<ul style="list-style-type: none"><li>• Specify the maximum (minimum) number of characters for a field</li></ul>
onClick="Value"	<ul style="list-style-type: none"><li>• Only used if the value is a button, specify an event to fire on click</li></ul>
value="Value"	<ul style="list-style-type: none"><li>• Specify an initial value for an input field</li></ul>
name="Value"	<ul style="list-style-type: none"><li>• Name tags (often used to distinguish certain elements in CSS and JavaScript)</li></ul>
id="Value"	<ul style="list-style-type: none"><li>• Specify the id of the tag (often used to distinguish certain elements in CSS and JavaScript)</li></ul>

# HTML - tags

## Understanding HTML with a lab exercise

- HTML key tags
  - Form-related tags
    - Key value types of the <input> tag type properties

Type	Description
hidden	<ul style="list-style-type: none"><li>• Value that contains information that is not visible to the user, but needs to be sent along with the request to the server.</li></ul>
text	<ul style="list-style-type: none"><li>• Text box where to enter text</li></ul>
password	<ul style="list-style-type: none"><li>• Password input field, masked on entry</li></ul>
checkbox	<ul style="list-style-type: none"><li>• Checkboxes to select (More than one can be selected.)</li></ul>
radio	<ul style="list-style-type: none"><li>• Radio button (only one can be selected.)</li></ul>
button	<ul style="list-style-type: none"><li>• General button</li></ul>
submit	<ul style="list-style-type: none"><li>• Send server button</li></ul>
reset	<ul style="list-style-type: none"><li>• Reset button</li></ul>

# HTML - tags

## Understanding HTML with a lab exercise

- HTML key tags
  - Lab exercise with form-related tags

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>HTML Tag Example</title>
 </head>
 <body>
 <form action="" method="post">
 ID : <input type="text" name="userid" id="userid">

 PW : <input type="password" name="pass" id="pass">

 Gender
 <input type="radio" name="gender" value="m">male
 <input type="radio" name="gender" value="f">female

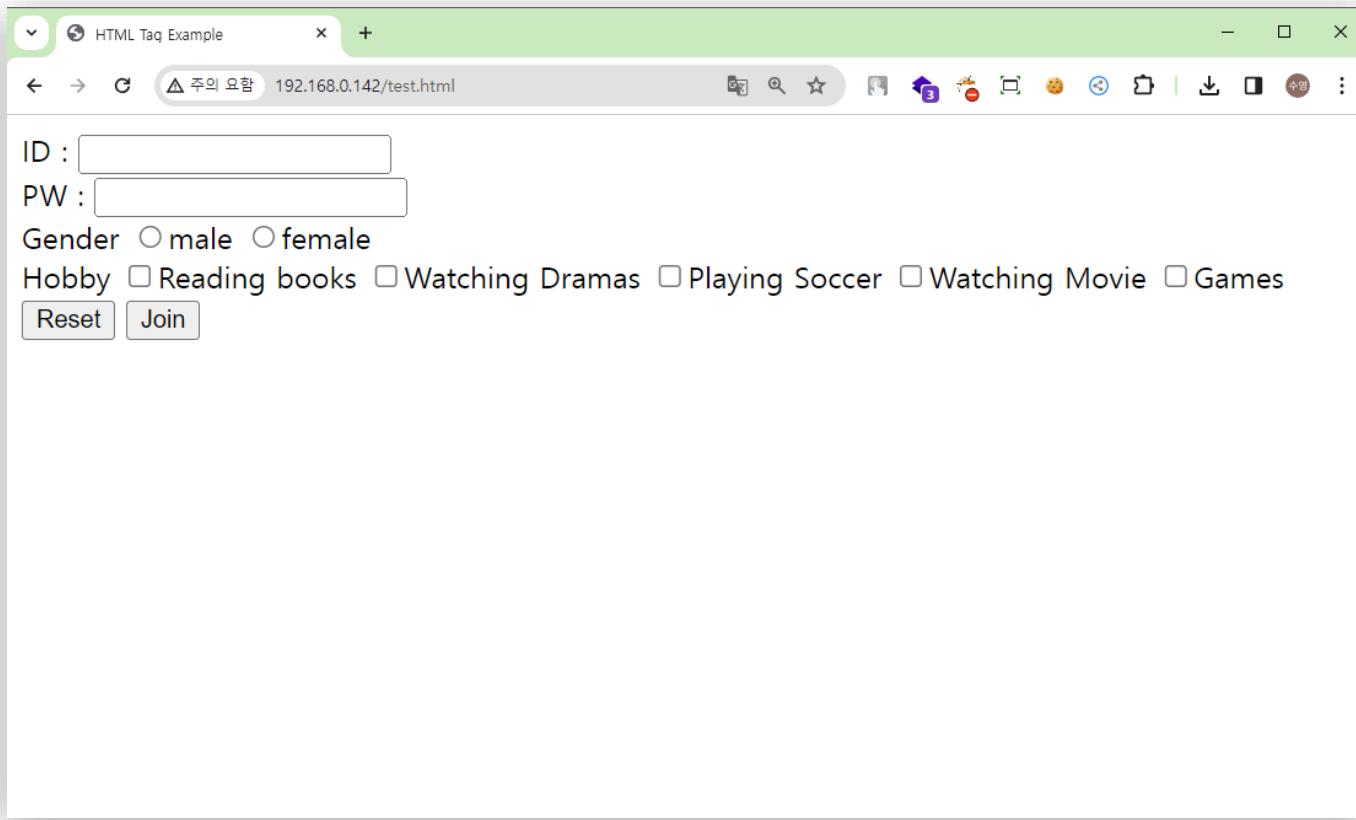
 Hobby
 <input type="checkbox" name="hobby" value="reading">Reading books
 <input type="checkbox" name="hobby" value="drama">Watching Dramas
 <input type="checkbox" name="hobby" value="soccer">Playing Soccer
 <input type="checkbox" name="hobby" value="movie">Watching Movie
 <input type="checkbox" name="hobby" value="game">Games

 <input type="reset" value="Reset"> <input type="submit" value="Join">
 </form>
 </body>
</html>
```

# HTML - tags

Understanding HTML with a lab exercise

- HTML key tags
  - Lab exercise with form-related tags
    - Save the file you created and check it by visiting <http://my ip/test.html> in a browser.



# JavaScript - basics

Understanding HTML with a lab exercise

JavaScript is an object-oriented script programming language. It is primarily used within web browsers and has the ability to access objects embedded in other applications. Along with HTML, it is one of the web components that make up the Web.

- JavaScript overview

- Created by Brendan Eich at Netscape Communication Corporation in 1995
- Developed under the name Mocha, later renamed LiveScript, and finally JavaScript.
- It is an object-based programming language, equivalent to a script language.
- A language unrelated to Java that runs on the JVM and is driven by a scripting engine (interpreter) inside the browser.
- Rather, it is heavily influenced by the C language and has a similar basic syntax.
  - Similar in the use of braced blocks, semicolons, variable declarations, operator usage, etc.
- Reference site
  - <http://tcpschool.com/javascript/intro>



# JavaScript - basics

Understanding HTML with a lab exercise

- How to load JavaScript

- Inline method

```
<!DOCTYPE html>
<html>
<body>
 <input type="button" onClick="alert('Hello world')" value="Hello world">
</body>
</html>
```

- <script> tag method

```
<!DOCTYPE html>
<html>
<body>
 <input type="button" id="acs" value="Hello world" />
 <script>
 var acs = document.getElementById('acs');
 hw.addEventListener('click',
 function() {
 alert('Hello world');
 }
)
 </script>
</body>
</html>
```

# JavaScript - basics

Understanding HTML with a lab exercise

- How to load JavaScript
  - External file method

```
<!DOCTYPE html>
<html>
<body>
 <input type="button" id="acs" value="Hello world" />
 <script src="acs.js"></script>
</body>
</html>
```

- acs.js

```
var acs = document.getElementById('acs');
hw.addEventListener('click',
 function(){
 alert('Hello world');
 }
)
```

# JavaScript - basics

## Understanding HTML with a lab exercise

- JavaScript base types

Type	Description
Numeric	<ul style="list-style-type: none"><li>• Express all numbers as a single real number, without distinguishing between integers and reals</li><li>• var num = 10;</li></ul>
String	<ul style="list-style-type: none"><li>• A set of characters enclosed in double ("") or single ( ' ' ) quotation marks.</li><li>• var myName = "Hong Gil-dong";</li></ul>
Undefined	<ul style="list-style-type: none"><li>• Means "type" is undefined</li><li>• var str;</li></ul>

- JavaScript variables

```
var month; : declaration of a variable
var date = 30; : initialize at the same time declare a variable
month = 12; : initialize a variable
year = 2020; : implicitly declare a variable named year
var month, date; : declare multiple variables at once
var hours = 7, minutes = 15; : initialize multiple variables simultaneously with declarations
month = 10, date = 5; : initialize multiple variables at once
```

# JavaScript - control statement

Understanding JavaScript with a lab exercise

- JavaScript conditional statements
  - if statements
    - Grammar

```
if(conditional statement1) {
 An executable statement you want to run when the result of conditional statement1 is true;
} else if(conditional statement2) {
 An executable statement you want to run when the result of conditional statement2 is true;
} else {
 An executable statement that you want to run when the result of conditional statement1 is
false, and the result of conditional statement2 is also false;
}
```

- Caveats
  - Incorrect grammar

```
if(x = y) {
 document.write("The two variables x and y are equal.");
}
```

- Correct grammar

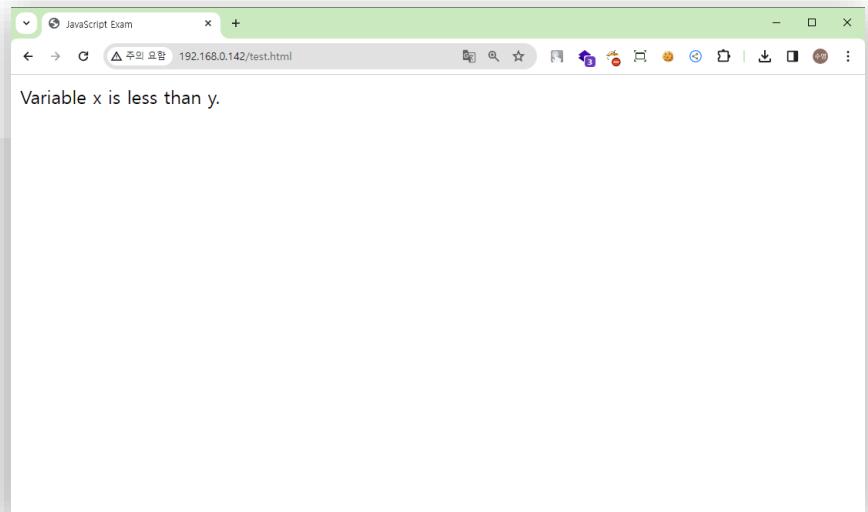
```
if(x == y) {
 document.write("The two variables x and y are equal.");
}
```

# JavaScript - control statement

Understanding JavaScript with a lab exercise

- JavaScript conditional statements
  - The if statement example

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
<head>
 <meta charset="UTF-8">
 <title>JavaScript Exam</title>
</head>
<body>
 <script>
 var x = 10, y = 20;
 if (x == y) {
 document.write("Variables x and y are equal.");
 } else if (x < y) {
 document.write("Variable x is less than y.");
 } else { // x > y인 경우
 document.write("Variable x is greater than y.");
 }
 </script>
</body>
</html>
```



# JavaScript - functions

## Understanding JavaScript with a lab exercise

- JavaScript functions
  - Function definition
    - Begin with a function keyword
    - Components
      - Function name
        - Identifier separating the function
      - Function parameters in parentheses, separated by commas (,)
        - Variable that makes the value passed as an argument when calling a function available within the function.
      - JavaScript statement enclosed in braces ({} )

```
function functionName(parameter1, parameter2, ...) {
 An executable statement you want to run when the function is called;
}
```

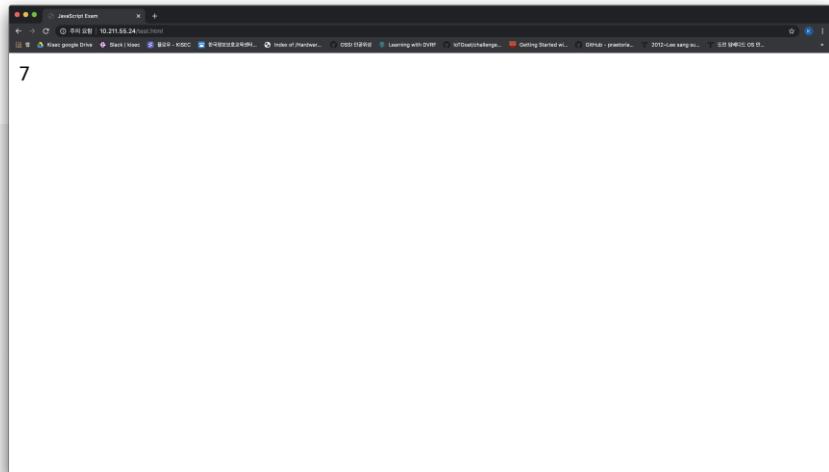
- Function calls
  - Available inside JavaScript or in HTML events (e.g., onClick event)

# JavaScript - functions

## Understanding JavaScript with a lab exercise

- JavaScript functions
  - Function usage example

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
<head>
 <meta charset="UTF-8">
 <title>JavaScript Exam</title>
</head>
<body>
 <script>
 function addNum(x, y) {
 return x + y;
 }
 var sum = addNum(4, 3);
 document.write(sum);
 </script>
</body>
</html>
```



# JavaScript - functions

## Understanding JavaScript with a lab exercise

- JavaScript objects
  - Object
    - Unordered set of properties consisting of a name and a value.
    - Function can also come as the value of a property, if that property is passed to the method
    - Create an object

```
var objectName = {
 property1Name : value of property1,
 property2Name : value of property2,
 ...
};
```

- Reference the properties of an object

```
objectName.property Name
or
objectName[“property Name”]
```

- Method reference of an object

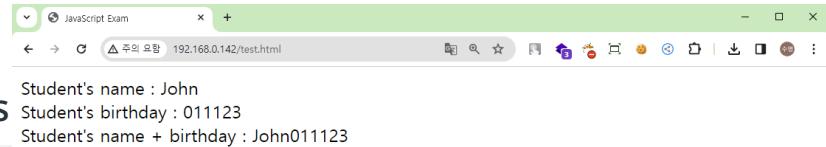
```
objectName.method Name()
```

# JavaScript - functions

## Understanding JavaScript with a lab exercise

- JavaScript objects
  - Lab exercise on creating and referencing objects

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
<head>
 <meta charset="UTF-8">
 <title>JavaScript Exam</title>
</head>
<body>
 <script>
 var student = {
 name: "John", // Define "name" property
 birthday: "011123", // Define "birthday" property
 full: function() { // Return "name" and "birthday" property
 return this.name + this.birthday;
 }
 };
 document.write("Student's name : " + student.name + "
");
 document.write("Student's birthday : " + student.birthday + "
");
 document.write(" Student's name + birthday : " + student.full());
 </script>
</body>
</html>
```



# JavaScript - functions

## Understanding JavaScript with a lab exercise

- JavaScript key usage features

- Show notifications

```
<script>
 function alertTest() {
 window.alert("Alert!!");
 }
</script>
```

- You can omit the window prefix for window objects
  - Move the page
    - Go to the previous page
    - Go to the specified page

```
<input type="button" value="back" onClick = history.go(-1)>
```

- Go to the specified page

```
<script>
 function testHref() {
 window.location.href='/test2.html';
 }
</script>
```

# JavaScript - functions

Understanding JavaScript with a lab exercise

- JavaScript key usage features
  - Replace HTML tag values
    - Separate <form> tags and <input> tags by the value of the name property
    - Change or inspect the property value inside that tag

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
<head>
 <meta charset="UTF-8">
 <title>JavaScript Exam</title>
</head>
<body>
 <form name="form_name">
 <input type="text" name="input_name" value="">
 <input type="button" value="Confirm" onClick=check()>
 </form>
 <script>
 function check() {
 if(form_name.input_name.value != "acs") form_name.input_name.value="acs";
 else form_name.input_name.value="Success!";
 }
 </script>
</body>
</html>
```

# JavaScript - functions

## Understanding JavaScript with a lab exercise

- JavaScript key usage features
  - Reposition the HTML tag cursor
    - Separate <form> tags and <input> tags by the value of the name property
    - Change the cursor position to the property value within that tag

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
<head>
 <meta charset="UTF-8">
 <title>JavaScript Exam</title>
</head>
<body>
 <form name="form_name">
 <input type="text" name="input1" value="">
 <input type="text" name="input2" value="">
 <input type="button" value="Confirm" onClick=check()
 </form>
 <script>
 function check() {
 form_name.input2.focus();
 }
 </script>
</body>
</html>
```

# JavaScript - implementing web page

Web fundamentals lab to implement a login page

- HTML login default form
  - Create a `<form>` tag, username `<input>` tag, password `<input>` tag, and login button `<input>` tag

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Login Form</title>
 </head>
 <body>
 <form>
 User ID : <input type=text value="">

 Password : <input type=password value="">

 <input type=submit value="Login">
 </form>
 </body>
</html>
```

# JavaScript - implementing web page

Web fundamentals lab to implement a login page

- HTML login default form
  - Set the <form> tag method property to get, name property to "Login\_form".
  - Set the <input> tag name properties to "id" and "passwd", respectively.

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Login Form</title>
 </head>
 <body>
 <form method="GET" name="Login_form">
 User ID : <input type=text name="id" value="">

 Password : <input type=password name="passwd" value="">

 <input type=submit value="Login">
 </form>
 </body>
</html>
```

# JavaScript - implementing web page

Web fundamentals lab to implement a login page

- Verify the JavaScript <input> tag string
  - Create a JavaScript check() function and set its onClick event to the submit button.
  - If any of the <input> tags id, passwd are empty, print "Id and password are invalid".

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Login Form</title>
 </head>
 <body>
 <form method="GET" name="Login_form">
 User ID : <input type=text name="id" value="">

 Password : <input type=password name="passwd" value="">

 <input type=submit value="Login" onClick=check()>
 </form>
 <script>
 function check() {
 if(Login_form.id.value == "" || Login_form.passwd.value == "") {
 alert("Invalid User ID or Password");
 }
 }
 </script>
 </body>
</html>
```

# JavaScript - implementing web page

Web fundamentals lab to implement a login page

- Verify the JavaScript <input> tag string
  - Remove all values that were entered after printing the phrase if the field is empty.

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
:
<body>
 <form method="GET" name="Login_form">
 User ID : <input type=text name="id" value="">

 Password : <input type=password name="passwd" value="">

 <input type=submit value="Login" onClick=check()>
 </form>
 <script>
 function check() {
 if(Login_form.id.value == "" || Login_form.passwd.value == "") {
 alert("Invalid User ID or Password");
 Login_form.id.value="";
 Login_form.passwd.value="";
 }
 }
 </script>
</body>
</html>
```

# JavaScript - implementing web page

Web fundamentals lab to implement a login page

- Verify the JavaScript <input> tag string

- Create a conditional statement of the check() function for the condition where id value is "acs" and passwd value is "acs123123" with the else if statement.

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
:
<body>
 <form method="GET" name="Login_form">
 User ID : <input type=text name="id" value="">

 Password : <input type=password name="passwd" value="">

 <input type=submit value="Login" onClick=check()
 </form>
 <script>
 function check() {
 if(Login_form.id.value == "" || Login_form.passwd.value == "") {
 alert("Invalid User ID or Password");
 Login_form.id.value="";
 Login_form.passwd.value="";
 }
 else if(Login_form.id.value == "acs" && Login_form.passwd.value == "acs123123") {
 }
 }
 </script>
</body>
</html>
```

# JavaScript - implementing web page

Web fundamentals lab to implement a login page

- Verify the JavaScript <input> tag string
  - If the else if condition is met, print "Login Success!" and go to login\_success.html in the same path.

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
<head>
<title>Login Page</title>
</head>
<body>
<h1>Login Page</h1>
<form name="Login_form">
<input type="text" name="id" placeholder="User ID" />
<input type="password" name="passwd" placeholder="Password" />

<input type="button" value="Login" />
</form>
<script>
function check() {
 if(Login_form.id.value == "" || Login_form.passwd.value == "") {
 alert("Invalid User ID or Password");
 Login_form.id.value="";
 Login_form.passwd.value="";
 }
 else if(Login_form.id.value == "acs" && Login_form.passwd.value == "acs123123") {
 window.location.href="/login_success.html";
 alert("Login Success!");
 }
 else
 alert(" Invalid User ID or Password");
}
</script>
</body>
</html>
```

# JavaScript - implementing web page

Web fundamentals lab to implement a login page

- Create a page to display when successful in logging in.

```
$ sudo vi /var/www/html/login_success.html
<!DOCTYPE html>
<html>
 <head>
 <title>Login Success</title>
 </head>
 <body>
 <h1>Login Success!!</h1>
 </body>
</html>
```

# JavaScript - implementing web page

Web fundamentals lab to implement a login page

- Screens for lab exercises

- Notification screen when login fails

A screenshot of a web browser window titled "Login Form". The URL bar shows "192.168.0.142/test.html?id=&pa...". The form has fields for "User ID" (acs) and "Password" (\*\*\*\*). A modal dialog box is displayed with the message "192.168.0.142 내용: Invalid User ID or Password" and a green "확인" (Confirm) button.

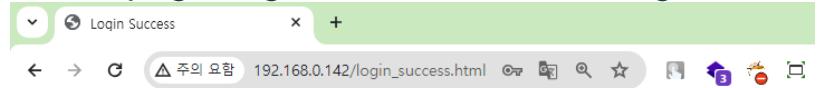
- If there is a blank space

A screenshot of a web browser window titled "Login Form". The URL bar shows "192.168.0.142/test.html?id=&pa...". The form has fields for "User ID" (acs) and "Password" (empty). A modal dialog box is displayed with the message "192.168.0.142 내용: Invalid User ID or Password" and a green "확인" (Confirm) button.

- Notification screen when login is successful

A screenshot of a web browser window titled "Login Form". The URL bar shows "192.168.0.142/test.html?id=acs&pa...". The form has fields for "User ID" (acs) and "Password" (\*\*\*\*\*). A modal dialog box is displayed with the message "192.168.0.142 내용: Login Success!" and a green "확인" (Confirm) button.

- The page to go to after successful login



# Login Success!!

# Understanding PHP language

## Understanding PHP with a lab exercise

PHP is a server-side script language that performs server-side processing and is based on the C language. When you add code written in PHP to HTML code, the web server interprets the code and automatically generates an HTML document.

- PHP: Hypertext Preprocessor (PHP) overview
  - First released in 1995, originally called Personal Home Page Tools, but later renamed
  - Designed to create dynamic web pages
    - Dynamic web page : a web page that is generated by processing data on the server with a server-side script language.
  - PHP features
    - Supported by all major operating systems and most web servers
    - Built for building simple sites, so it's inefficient for building complex sites.
    - More intuitive to code than other programming languages
    - Specialized in text processing, suitable for processing HTML documents
    - Insecure language constructs



# Understanding PHP language

## Understanding PHP with a lab exercise

PHP is a server-side script language that performs server-side processing and is based on the C language. When you add code written in PHP to HTML code, the web server interprets the code and automatically generates an HTML document.

- PHP: Hypertext Preprocessor (PHP) overview
  - Compare the major web programming languages

Language	OS	Primary web server	Manufacturer	Speed	Portability
PHP	Linux, Windows	Apache	Open Source	Fast	Average
JSP	Linux, Windows	Tomcat	Oracle	Slow	Good
ASP.NET	Windows	IIS	Microsoft	Average	Bad

- Choose the right language based on your purpose and context
- Typical programs implemented in PHP
  - WordPress
  - gnuboard
  - MediaWiki

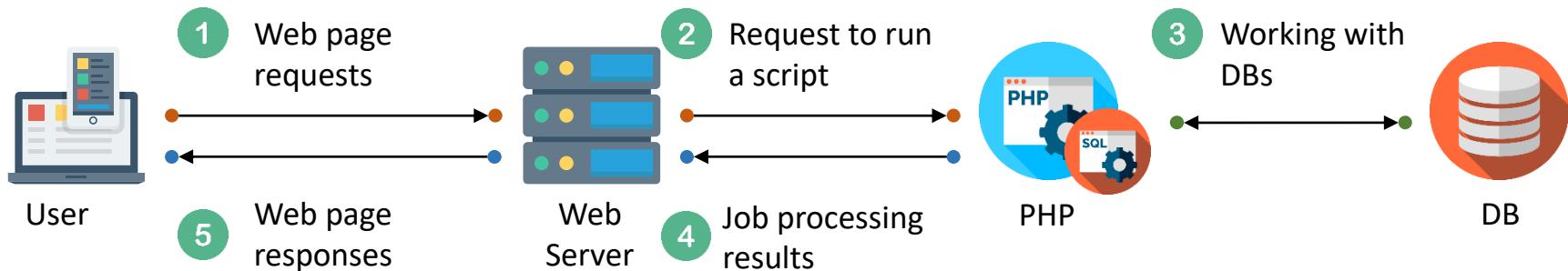
# Understanding PHP language

Understanding PHP with a lab exercise

PHP is a server-side script language that performs server-side processing and is based on the C language. When you add code written in PHP to HTML code, the web server interprets the code and automatically generates an HTML document.

- PHP: Hypertext Preprocessor (PHP) overview

- How PHP works



- A user requests a desired page from a web server via a web browser.
- The web server asks PHP for processing to render the page the user requested.
- PHP does data processing in conjunction with the DB if necessary.
- PHP passes the results of its work to the web server
- The web server completes the web page with the data it received and sends a response to the web browser.

# PHP - basic syntax

## Understanding PHP with a lab exercise

PHP is a server-side script language that performs server-side processing and is based on the C language. When you add code written in PHP to HTML code, the web server interprets the code and automatically generates an HTML document.

- PHP basic syntax
  - Set up PHP code areas
    - PHP code can be placed anywhere on a web page, with its start and end marked

PHP recommended style : <?php ... ?>

When short\_open\_tag is enabled : <? ... ?>

- Enable the short\_open\_tag feature
  - Change the feature from "Off" to "On" in the /etc/php.ini file, the PHP configuration file.

```
$ sudo vi /etc/php.ini
:
:
short_open_tag = Off # Change to On
:
:
```

# PHP - basic syntax

Understanding PHP with a lab exercise

- PHP basic syntax
  - PHP syntax
    - As in C, PHP commands end with a semicolon (;).
    - A line automatically ends without a semicolon, but the default usage is recommended

```
<?php
echo "Hello World";
?>
```

- PHP comments
  - Used for the reference by other developers, and for debugging in web development.
  - You can nest single-line comments within multi-line comments, but not between multi-line comments.

```
// Comments

/*
 Multi-line comments
*/

Comments
```

# PHP - basic syntax

## Understanding PHP with a lab exercise

- PHP basic syntax
  - PHP Variables
    - Declare a variable

```
$variableName = value;
```

- Prefix a variable name with a dollar sign (\$)
- Unlike C, you don't specify a type when you declare a variable; it is automatically determined by the value you assign to it.

```
$value = 7; // substitute integer value
$value = 12.3; // substitute a real number value
$value = "ACS"; // substitute string
```

- Rules for creating variable names
  - Case-sensitive, consisting only of alphabetic characters, numbers, and an underscore (\_).
  - A number cannot be the first character of a variable name.
  - Variable names cannot contain spaces.
  - Cannot use the internally predefined \$this.

# PHP - basic syntax

## Understanding PHP with a lab exercise

- PHP basic syntax
  - PHP form processing
    - Actions for processing data sent via HTML form tags

```
GET method : $value = $_GET["value"];
POST method : $value = $_POST["value"];
```

- GET method
  - Add data to an address for delivery

```
http://xxx.xxx.xxx.xxx/test.php?value=value
```

- POST method
  - Deliver data as a separate attachment
  - The POST method is recommended because it is more secure than the GET method.

```
http://xxx.xxx.xxx.xxx/test.php
```

# PHP - basic syntax

## Understanding PHP with a lab exercise

- PHP basic syntax
  - PHP form processing example

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
<body>
 <form action="output.php" method="post">
 Name : <input type="text" name="name">

 Age : <input type="text" name="age">

 <input type="submit" value="Confirm">
 </form>
</body>
</html>
```

```
$ sudo vi /var/www/html/output.php
<?php
 $name = $_POST["name"];
 $age = $_POST["age"];

 echo "Name is ", $name, ", and Age is ", $age;
?>
```

# PHP - basic syntax

Understanding PHP with a lab exercise

- PHP basic syntax
  - PHP form processing example

192.168.0.142/test.html

Name :

Age :

Confirm

192.168.0.142/output.php

Name is acs, and Age is 25

# PHP - basic syntax

Understanding PHP with a lab exercise

- PHP and JavaScript
  - Use JavaScript within a PHP page

```
echo "<script> String </script>";
```

- Example

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
 <body>
 <form method=post name=Login_form action="/login_check.php">
 User ID : <input type=text name="id" value="">

 Password : <input type=password name="passwd" value="">

 <input type=submit value="Login" onClick = check()
 </form>
 <script>
 function check() {
 if(Login_form.id.value == "" || Login_form.passwd.value == "") {
 alert("Invalid User ID or Password");
 Login_form.id.value="";
 Login_form.passwd.value="";
 }
 }
 </script>
 </body>
</html>
```

# PHP - basic syntax

Understanding PHP with a lab exercise

- PHP and JavaScript
  - Example

```
$ sudo vi /var/www/html/login_check.php
<?php
 $id=$_POST['id'];
 $pw=$_POST['passwd'];
 if($id == "test" && $pw == "1234") {
 echo "<script>
 alert(\"Login Success!\");
 window.location.href='/login_success.html';
 </script>
 ";
 }
 else {
 echo "<script>
 alert(\"Login Failed!\");
 window.location.href='/test.html';
 </script>
 ";
}
?>
```

# PHP - basic syntax

Understanding PHP with a lab exercise

- PHP and JavaScript
  - Example

The screenshot shows a web browser with two tabs and a modal dialog.

- Left Tab:** Title: 192.168.0.142/test.html. It contains a login form with fields for "User ID" (test) and "Password" (\*\*\*\*), and a "Login" button.
- Right Tab:** Title: Login Success. It displays the message "Login Success!!".
- Bottom Modal:** Title: 192.168.0.142 내용:. It contains the text "Login Success!" and a green "확인" (Confirm) button.

# PHP - DB connection

## Understanding PHP with a lab exercise

- PHP and MariaDB
  - PHP & MariaDB functions
    - Connection functions

```
$conn = mysqli_connect($server_name, $user_name, $password, $database_name);
```

- Disconnect functions

```
mysqli_close($conn);
```

- Statement execution functions

```
$result = mysqli_query($conn, $query);
```

- Return the number of rows resulting from a query

```
mysqli_num_rows($result);
```

- Function to store query results in an array with column names as array indices

```
$row = mysqli_fetch_array($result);
```

- The \$row variable has its value available as \$row['column name'].
- Memory release function on query statement results, used before the mysqli\_close function

```
mysqli_free_result($result);
```

# PHP - DB connection

## Understanding PHP with a lab exercise

- PHP and MariaDB
  - PHP & MariaDB function examples
    - Create a MariaDB database with tables

```
$ mysql -u root -p
Enter password: // Enter password
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 124
Server version: 5.5.64-MariaDB MariaDB Server
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> create database ACS;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| information_schema |
| ACS |
| mysql |
| performance_schema |
+-----+
4 rows in set (0.00 sec)
```

# PHP - DB connection

## Understanding PHP with a lab exercise

- PHP and MariaDB
  - PHP & MariaDB function examples
    - Create a MariaDB database with tables

```
MariaDB [(none)]> use ACS;
Database changed
MariaDB [ACS]> create table login (idx int not null auto_increment primary key,
-> id varchar(20), pw varchar(20));
Query OK, 0 rows affected (0.01 sec)
MariaDB [ACS]> show tables;
+-----+
| Tables_in_ACS |
+-----+
| login |
+-----+
1 rows in set (0.00 sec)
MariaDB [ACS]> insert into login (id, pw) values ("acs", "acs123123"), ("admin", "toor"); // Insert example
Query OK, 2 rows affected (0.00 sec)
MariaDB [ACS]> select * from login;
+----+----+----+
| idx | id | pw |
+----+----+----+
| 1 | acs | acs123123 |
| 2 | admin | toor |
+----+----+----+
2 rows in set (0.00 sec)
```

# PHP - DB connection

## Understanding PHP with a lab exercise

- PHP and MariaDB

- PHP & MariaDB function examples

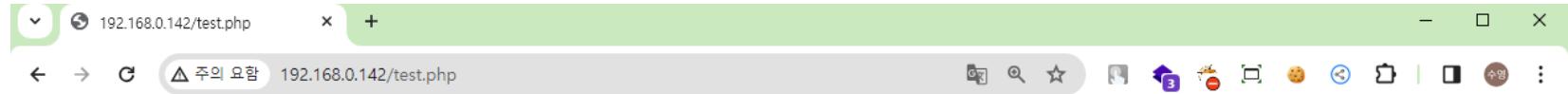
```
$ sudo vi /var/www/html/test.php
<?php
 $conn = mysqli_connect("localhost", "root", "acs123123", "ACS");
 if(!$conn) {
 echo "Error Unable to connect to Mariadb.";
 exit;
 }

 $query = "SELECT * FROM login";
 $result = mysqli_query($conn, $query);
 $re_cnt = mysqli_num_rows($result);
 if($re_cnt > 0) {
 while($row = mysqli_fetch_array($result)) {
 echo "Num .", $row['idx'], "
ID : ", $row['id'], "
Password : ", $row['pw'], "
";
 }
 }
 else {
 echo "Not Found Result!";
 }
 mysqli_free_result($result);
 mysqli_close($conn);
?>
```

# PHP - DB connection

Understanding PHP with a lab exercise

- PHP and MariaDB
  - PHP & MariaDB function examples



# PHP - DB connection

## Web application lab exercise to implement a login page

- PHP and MariaDB

- Automatic connection by including files when linking DB

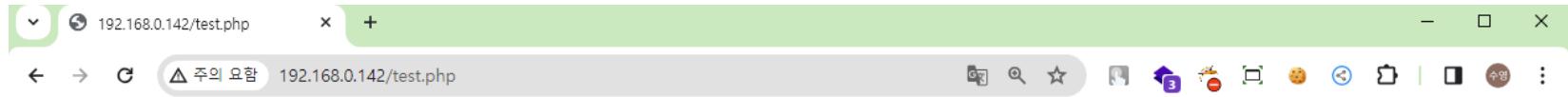
```
$ sudo vi /var/www/html/db_conn.php
<?php
 $conn = mysqli_connect("localhost", "root", "acs123123", "ACS");
 if(!$conn) {
 echo "Error Unable to connect to Maraidb.";
 exit;
 }
 mysqli_query($conn, "SET NAMES utf8"); // Handle encoding when inserting Korean into DB
?>
```

```
$ sudo vi /var/www/html/test.php
<?php
 include "db_conn.php";
 $query = "SELECT * FROM login";
 $result = mysqli_query($conn, $query);
 $re_cnt = mysqli_num_rows($result);
 if($re_cnt > 0) {
 while($row = mysqli_fetch_array($result)) {
 echo "Num .", $row['idx'], "
ID : ", $row['id'], "
Password : ", $row['pw'], "
";
 }
 }
 else echo "Not Found Result!";
 mysqli_free_result($result);
 mysqli_close($conn);
?>
```

# PHP - DB connection

Web application lab exercise to implement a login page

- PHP and MariaDB
  - Automatic connection by including files when linking DB



# PHP - implementing login page

Web application lab exercise to implement a login page

- HTML login basic form for PHP
  - Add the option to pass the data from the <form> tag to the PHP file that actually verifies the username and password.

```
$ sudo vi /var/www/html/test.html
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Login Form</title>
 </head>
 <body>
 <form method="GET" name="Login_form" action="/login_check.php">
 User ID : <input type=text name="id" value="">

 Password : <input type=password name="passwd" value="">

 <input type=submit value="Login" onClick=check()
 </form>
 <script>
 function check() {
 if(Login_form.id.value == "" || Login_form.passwd.value == "") {
 alert("Invalid User ID or Password");
 Login_form.id.value="";
 Login_form.passwd.value="";
 }
 }
 </script>
 </body>
</html>
```

# PHP - implementing login page

Web application lab exercise to implement a login page

- PHP for verifying usernames and passwords
  - Create a file that connects the passed data to MariaDB to process the login.
    - Store the information passed through the GET method as a PHP variable.
    - Connect to the DB and execute a SELECT statement using the ID as a condition.
    - Store in array when found.

```
$ sudo vi /var/www/html/login_check.php
<?php
 $id=$_GET['id'];
 $pw=$_GET['passwd'];

 include "db_conn.php";
 $query = "SELECT * FROM login WHERE id='$id'";
 $result = mysqli_query($conn, $query);
 $re_cnt = mysqli_num_rows($result);
 if($re_cnt > 0) {
 $row = mysqli_fetch_array($result);
 }
 mysqli_free_result($result);
 mysqli_close($conn);
?>
```

# PHP - implementing login page

Web application lab exercise to implement a login page

- PHP for verifying usernames and passwords
  - Create a file that connects the passed data to MariaDB to process the login.
    - Redirect to the login\_success.html file after comparing the information stored in the \$pw variable with the information in the DB.

```
$ sudo vi /var/www/html/login_check.php
<?php
 $id=$_GET['id'];
 $pw=$_GET['passwd'];

 include "db_conn.php";
 $query = "SELECT * FROM login WHERE id='$id'";
 $result = mysqli_query($conn, $query);
 $re_cnt = mysqli_num_rows($result);
 if($re_cnt > 0) {
 $row = mysqli_fetch_array($result);
 if($row['pw'] == $pw) {
 mysqli_free_result($result);
 mysqli_close($conn);
 echo "<script> window.location.href='/login_success.html'; </script>";
 }
 }
 mysqli_free_result($result);
 mysqli_close($conn);
?>
```

# PHP - implementing login page

Web application lab exercise to implement a login page

- PHP for verifying usernames and passwords
  - Create a file that connects the passed data to MariaDB to process the login.
    - Create a message to be displayed and return to the main page if the login fails because the conditions in the if statement are not met.

```
$ sudo vi /var/www/html/login_check.php
<?php
:
if($re_cnt > 0) {
 $row = mysqli_fetch_array($result);
 if($row['pw'] == $pw) {
 mysqli_free_result($result);
 mysqli_close($conn);
 echo "<script> window.location.href='/login_success.html'; </script>";
 }
}
mysqli_free_result($result);
mysqli_close($conn);
echo "<script>
 alert(\"Login Failed! Check Your ID and Password!!\");
 window.location.href='/test.html';
</script>
";
?>
```

# PHP - implementing login page

Web application lab exercise to implement a login page

- PHP for verifying usernames and passwords
  - Create a file that connects the passed data to MariaDB to process the login.
    - Redirect to the default page if an empty value is passed to the GET method.
      - PHP empty(\$value) function
        - Check if the variable is empty and return false if it is not.

```
$ sudo vi /var/www/html/login_check.php
<?php
 $id=$_GET['id'];
 $pw=$_GET['passwd'];

 if(empty($id) || empty($pw)) {
 echo "<script> window.location.href='/test.html'; </script>";
 }

 include "db_conn.php";

 $query = "SELECT * FROM login WHERE id='$id'";
 $result = mysqli_query($conn, $query);
 $re_cnt = mysqli_num_rows($result);
 :
?>
```

# PHP - implementing login page

Web application lab exercise to implement a login page

- Complete PHP source for username and password verification

```
<?php
 $id=$_GET['id'];
 $pw=$_GET['passwd'];

 if(empty($id) || empty($pw)) {
 echo "<script> window.location.href='/test.html'; </script>";
 }

 include "db_conn.php";

 $query = "SELECT * FROM login WHERE id='$id'";
 $result = mysqli_query($conn, $query);
 $re_cnt = mysqli_num_rows($result);
 if($re_cnt > 0) {
 $row = mysqli_fetch_array($result);
 if($row['pw'] == $pw) {
 mysqli_free_result($result);
 mysqli_close($conn);
 echo "<script> window.location.href='/login_success.html'; </script>";
 }
 }
 mysqli_free_result($result);
 mysqli_close($conn);
 echo "<script>
 alert(\"Login Failed! Check Your ID and Password!!\");
 window.location.href='/test.html';
 </script>
";
?>
```

# PHP - implementing login page

Web application lab exercise to implement a login page

- Screens for lab exercises

- Main start screen

User ID :

Password :

Login

- If there is a blank space

User ID : test

Password :

Login

- If your password or username is incorrect

192.168.0.142 내용:  
Login Failed! Check Your ID and Password!!

확인

- Page moved after successful login

192.168.0.142 주의 요함 /login\_success.html

# Login Success!!

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Create a DB table for the board

```
$ mysql -u root -p
Enter password: // Enter password
MariaDB [(none)]> use ACS;
Database changed
MariaDB [(none)]> create table board (
idx int not null auto_increment primary key,
name varchar(20) not null,
email varchar(30) not null,
pw varchar(12) not null,
title varchar(100) not null,
content text not null,
wdate TIMESTAMP DEFAULT NOW());
Query OK, 0 rows affected (0.01 sec)
MariaDB [(none)]>desc board;
+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+
| idx | int(11) | NO | PRI | NULL | auto_increment |
| name | varchar(20)| NO | | NULL |
| email | varchar(30)| NO | | NULL |
| pw | varchar(12)| NO | | NULL |
| title | varchar(100)| NO | | NULL |
| content | text | NO | | NULL |
| wdate | timestamp| NO | | CURRENT_TIMESTAMP |
+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Implement a board posting page
  - Create a form for entering name, email, password, title, and content
    - Define a name value for each  tag that sends data from this form to input\_write.php
    - The  tag has a default size of 10 lines and 50 spaces.
    - Inside the  tags, there are two buttons : a (regular) button (to go back to the previous page after canceling) and a submit button.

```
$ sudo vi /var/www/html/write.php
<!DOCTYPE HTML>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Write Board</title>
 </head>
 <body>
 <form action="/input_write.php" method="post">
 Name : <input type="text" name="name">

 E-Mail : <input type="text" name="email">

 Password : <input type="password" name="passwd">

 Title : <input type="text" name="title">

 Contents : <textarea rows="10" cols="50" name="content" ></textarea>

 <input type="button" value="Cancel" onClick = history.go(-1)><input type="submit" value="Save">
 </form>
 </body>
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Implement a board posting page

The screenshot shows a web browser window with the title "Write Board". The address bar displays the URL "192.168.0.142/write.php". The page content is a form for posting a new board entry. It includes fields for Name, E-Mail, Password, Title, and a large area for Contents, followed by "Cancel" and "Save" buttons.

Name :

E-Mail :

Password :

Title :

Contents :

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process data for the board posting page
  - Store data passed through the POST method in each variable
  - Save to DB with insert statement and return to list with success message

```
$ sudo vi /var/www/html/input_write.php
<?php
 include "db_conn.php";

 $name = $_POST['name'];
 $email = $_POST['email'];
 $passwd = $_POST['passwd'];
 $title = $_POST['title'];
 $content = $_POST['content'];

 $query = "insert into board (name, email, pw, title, content) values ('$name', '$email', '$passwd',
'$title', '$content')";
 $result = mysqli_query($conn, $query);
 if($result) {
 echo "<script> alert(\"Write Success!!!!\"); </script>";
 }
 else {
 echo "<script> alert(\"Board Write Error!\"); </script>";
 }
 mysqli_free_result($result);
 mysqli_close($conn);
 echo "<script> window.location.href='/list.php'; </script>";
?
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process data for the board posting page

Name :

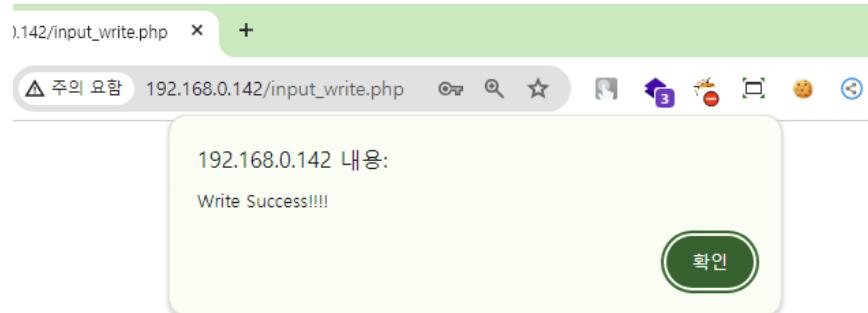
E-Mail :

Password :

Title :

Contents :

First Value



# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process a board post list - 1
  - Create a bulletin board list HTML table
  - The <thead> part is going to be static in the list, so write it ahead of time.

```
$ sudo vi /var/www/html/list.php
<!DOCTYPE HTML>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Board List</title>
 </head>
 <body>
 <table border=1>
 <caption>Board List</caption>
 <thead>
 <tr>
 <th>Num</th>
 <th>Author</th>
 <th>Title</th>
 <th>Date</th>
 </tr>
 </thead>
 </table>
 </body>
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process a board post list - 2
  - Create a bulletin board list HTML table
  - Create a <tbody> section that will be written repeatedly based on the number of posts

```
$ sudo vi /var/www/html/list.php
<!DOCTYPE HTML>
<html>
:
<thead>
 <tr>
 <th>Num</th>
 <th>Author</th>
 <th>Title</th>
 <th>Date</th>
 </tr>
</thead>
<tbody>
 <tr>
 <td></td>
 <td></td>
 <td></td>
 <td></td>
 </tr>
</tbody>
:
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process a board post list - 3
  - Connect to DB and search for board table data needed for current board list in descending order

```
$ sudo vi /var/www/html/list.php
<!DOCTYPE HTML>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Board List</title>
 </head>
 <body>
 <?php
 include "db_conn.php";
 $query = "select idx, name, title, wdate from board order by idx desc";
 $result = mysqli_query($conn, $query);
 ?>
 <table border=1>
 <caption>Board List</caption>
 <thead>
 <tr>
 <th>Num</th>
 <th>Author</th>
 :
 :
 </thead>
 <tbody>
 <tr>
 <td>1</td>
 <td>John Doe</td>
 </tr>
 <tr>
 <td>2</td>
 <td>Jane Smith</td>
 </tr>
 <tr>
 <td>3</td>
 <td>Bob Johnson</td>
 </tr>
 <tr>
 <td>4</td>
 <td>Alice Williams</td>
 </tr>
 <tr>
 <td>5</td>
 <td>David Lee</td>
 </tr>
 <tr>
 <td>6</td>
 <td>Sarah Brown</td>
 </tr>
 <tr>
 <td>7</td>
 <td>Michael Green</td>
 </tr>
 <tr>
 <td>8</td>
 <td>Laura Blue</td>
 </tr>
 <tr>
 <td>9</td>
 <td>Kevin Red</td>
 </tr>
 <tr>
 <td>10</td>
 <td>Emily Black</td>
 </tr>
 </tbody>
 </table>
 </body>
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process a board post list - 4
  - Output the text that will be printed in the <tbody> part to fit the <thead> part, respectively

```
$ sudo vi /var/www/html/list.php
<!DOCTYPE HTML>
<html>
:
<tbody>
<?php
$re_cnt = mysqli_num_rows($result);
if($re_cnt > 0) {
 while($row = mysqli_fetch_array($result)) {
?
 <tr>
 <td><?php echo $row['idx']; ?></td>
 <td><?php echo $row['name']; ?></td>
 <td><?php echo $row['title']; ?></td>
 <td><?php echo $row['wdate']; ?></td>
 </tr>
<?php
 }
}
mysqli_free_result($result);
mysqli_close($conn);
?>
</tbody>
:
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process a board post list - 5
  - Set up an  tag that passes the number to a page where you can view the content of the post, which will be implemented later

```
$ sudo vi /var/www/html/list.php
<!DOCTYPE HTML>
<html>
:
 <tbody>
<?php
$re_cnt = mysqli_num_rows($result);
if($re_cnt > 0) {
 while($row = mysqli_fetch_array($result)) {
?>
 <tr>
 <td><?php echo $row['idx']; ?></td>
 <td><?php echo $row['name']; ?></td>
 <td><a href="view.php?num=<?php echo $row['idx']; ?>"><?php echo $row['title']; ?></td>
 <td><?php echo $row['wdate']; ?></td>
 </tr>
<?php
 }
}
mysqli_free_result($result);
mysqli_close($conn);
?>
 </tbody>
:
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process a board post list - 6
  - Add a write button

```
$ sudo vi /var/www/html/list.php
<!DOCTYPE HTML>
<html>
:
<?php
 }
}
mysqli_free_result($result);
mysqli_close($conn);
?>
 </tbody>
</table>
<input type="button" value="Write" onClick = go_write()
<script>
 function go_write() {
 window.location.href='/write.php';
 }
</script>
</body>
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process a board post list

Board List

Num	Author	Title	Date
2	acs	<a href="#">Second</a>	2024-01-08 17:57:37
1	acs	<a href="#">First</a>	2024-01-08 17:48:24

Write

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Complete source code of the board post list - 1

```
<!DOCTYPE HTML>
<html>
<!DOCTYPE HTML>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Board List</title>
 </head>
 <body>
 <?php
 include "db_conn.php";
 $query = "select idx, name, title, wdate from board order by idx desc";
 $result = mysqli_query($conn, $query);
 ?>
 <table border=1>
 <caption>Board List</caption>
 <thead>
 <tr>
 <th>Num</th>
 <th>Author</th>
 <th>Title</th>
 <th>Date</th>
 </tr>
 </thead>
 <tbody>
 <tr>
 <td>1</td>
 <td>John Doe</td>
 <td>Learn PHP</td>
 <td>2023-10-01</td>
 </tr>
 <tr>
 <td>2</td>
 <td>Jane Smith</td>
 <td>Learn MySQL</td>
 <td>2023-10-02</td>
 </tr>
 <tr>
 <td>3</td>
 <td>Bob Johnson</td>
 <td>Learn Java</td>
 <td>2023-10-03</td>
 </tr>
 </tbody>
 </table>
 </body>
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Complete source code of the board post list - 2

```
:
 <tbody>
 <?php
 $re_cnt = mysqli_num_rows($result);
 if($re_cnt > 0) {
 while($row = mysqli_fetch_array($result)) {
 ?>
 <tr>
 <td><?php echo $row['idx']; ?></td>
 <td><?php echo $row['name']; ?></td>
 <td><a href="view.php?num=<?php echo $row['idx']; ?>"><?php echo
$row['title']; ?></td>
 <td><?php echo $row['wdate']; ?></td>
 </tr>
 <?php
 }
 }
 mysqli_free_result($result);
 mysqli_close($conn);
 ?>
 </tbody>
 </table>
 <input type="button" value="Write" onClick = go_write()>
 <script>
 function go_write() {
 window.location.href='/write.php';
 }
 </script>
 </body>
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Implement a page to view a board post - 1
  - Implement the base form of the post view table HTML tag

```
$ sudo vi /var/www/html/view.php
:
<body>
 <table>
 <tr>
 <td>Title</td>
 </tr>
 <tr>
 <td>Name</td>
 </tr>
 <tr>
 <td>E-Mail</td>
 </tr>
 <tr>
 <td>Date</td>
 </tr>
 <tr>
 <td>Contents</td>
 </tr>
 </table>
</body>
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Implement a page to view a board post - 2
  - Implement buttons for list, modify post, and delete post  
(modify post : modify.php, delete post: delete.php)
    - Edited and deleted posts will be sent with their post numbers, as edits and deletions should be based on those post numbers in the future.

```
$ sudo vi /var/www/html/view.php
:
<tr>
 <td>Name</td>
</tr>
<tr>
 <td>E-Mail</td>
</tr>
<tr>
 <td>Date</td>
</tr>
<tr>
 <td>Contents</td>
</tr>
</table>
[List]
<a href="/modify.php?num=<?php echo $_GET['num']; ?>">[Edit]
<a href="/delete.php?num=<?php echo $_GET['num']; ?>">[Delete]
</body>
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Implement a page to view a board post - 3
  - Retrieve data needed to display post content
    - Output subject, name, email, date, and content

```
$ sudo vi /var/www/html/view.php
:
<body>
 <?php
 include "db_conn.php";
 $query = "select title, name, email, wdate, content from board where idx=" . $_GET['num'];
 $result = mysqli_query($conn, $query);
 $re_cnt = mysqli_num_rows($result);
 if($re_cnt > 0) {
 $row = mysqli_fetch_array($result);
 }
 ?>
 <table>
 :
 </table>
 <?php
 }
 mysqli_free_result($result);
 mysqli_close($conn);
 ?>
 :
 </body>
 </html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

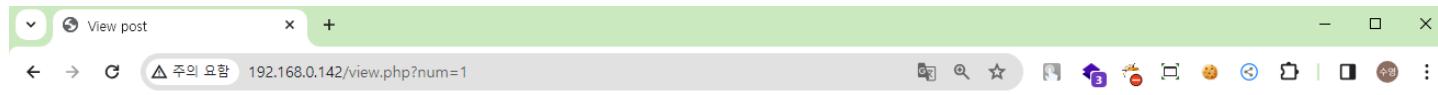
- Implement a page to view a board post - 4
  - Retrieve the data needed to display the post content and output it for each field.

```
$ sudo vi /var/www/html/view.php
:
<table>
 <tr>
 <td>Title</td>
 <td><?php echo $row['title'] ?></td>
 </tr>
 <tr>
 <td>Name</td>
 <td><?php echo $row['name'] ?></td>
 </tr>
 <tr>
 <td>E-Mail</td>
 <td><?php echo $row['email'] ?></td>
 </tr>
 <tr>
 <td>Date</td>
 <td><?php echo $row['wdate'] ?></td>
 </tr>
 <tr>
 <td>Contents</td>
 <td><?php echo $row['content'] ?></td>
 </tr>
</table>
:
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Implement a page to view a board post



Title      First

Name      acs

E-Mail    acs@acs.com

Date      2024-01-08 17:48:24

Contents First Value

[\[List\]](#) [\[Edit\]](#) [\[Delete\]](#)

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Complete source code of the page to view a board post - 1

```
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>View post</title>
 </head>
 <body>
 <?php
 include "db_conn.php";
 $query = "select title, name, email, wdate, content from board where idx=" . $_GET['num'];
 $result = mysqli_query($conn, $query);
 $re_cnt = mysqli_num_rows($result);
 if($re_cnt > 0) {
 $row = mysqli_fetch_array($result);
 }
 ?>
 <table>
 <tr>
 <td>Title</td>
 <td><?php echo $row['title'] ?></td>
 </tr>
 <tr>
 <td>Name</td>
 <td><?php echo $row['name'] ?></td>
 </tr>
 :
 :
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Complete source code of the page to view a board post - 2

```
<tr>
 <td>E-Mail</td>
 <td><?php echo $row['email'] ?></td>
</tr>
<tr>
 <td>Date</td>
 <td><?php echo $row['wdate'] ?></td>
</tr>
<tr>
 <td>Contents</td>
 <td><?php echo $row['content'] ?></td>
</tr>
</table>
<?php
}
mysqli_free_result($result);
mysqli_close($conn);
?>
[List]
<a href="/modify.php?num=<?php echo $_GET['num']; ?>">[Edit]
<a href="/delete.php?num=<?php echo $_GET['num']; ?>">[Delete]
</body>
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Implement a page to edit a board post - 1
  - In the existing board post view page source code, wrap the `<table>` tag with a `<form>` tag, and output the title and content within the `<input>` tag `text` property and `<textarea>` tag, respectively.
  - Send a password field for identification on changes, along with the `hidden` property of the `<input>` tag to pass the number
  - Implement a back button to cancel edits and return to the view page outside of the `<form>` tag

```
$ sudo vi /var/www/html/modify.php
<!DOCTYPE html>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Edit Post</title>
 </head>
 <body>
 <?php
 include "db_conn.php";
 $idx = $_GET['num'];
 $query = "select title, name, email, wdate, content, pw from board where idx=$idx";
 $result = mysqli_query($conn, $query);
 $re_cnt = mysqli_num_rows($result);
 if($re_cnt > 0) {
 $row = mysqli_fetch_array($result);
 }
 </?php>
 </body>
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Implement a page to edit a board post - 2

```
:
?>
<form action="/input_modify.php" method="post">
 <table>
 <tr>
 <td>Title</td>
 <td><input type="text" name="title" value="<?php echo $row['title'] ?>"></td>
 </tr>
 <tr>
 <td>Name</td>
 <td><?php echo $row['name'] ?></td>
 </tr>
 <tr>
 <td>E-Mail</td>
 <td><?php echo $row['email'] ?></td>
 </tr>
 <tr>
 <td>Date</td>
 <td><?php echo $row['wdate'] ?></td>
 </tr>
 <tr>
 <td>Contents</td>
 <td><textarea rows="10" cols="50" name="content"><?php echo
$row['content'] ?></textarea></td>
 </tr>
:
:
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Implement a page to edit a board post - 3

```
:
 <tr>
 <td>Password</td>
 <td><input type="password" name="passwd"></td>
 </tr>
 </table>
 <input type="hidden" name="num" value="<?php echo $idx ?>">
 <input type="submit" value="Save">
 </form>
<?php
 }
 mysqli_free_result($result);
 mysqli_close($conn);
?>
 <input type="button" value="[Back]" onClick = history.go(-1)>
 </body>
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Implement a page to edit a board post

The screenshot shows a web browser window titled "Edit Post". The address bar displays the URL "192.168.0.142/modify.php?num=1". The page content is a form for editing a post. The form fields are as follows:

Title	<input type="text" value="First"/>
Name	acs
E-Mail	acs@acs.com
Date	2024-01-08 17:48:24
Contents	<div style="border: 1px solid black; height: 200px; width: 100%;">First Value</div>
Password	<input type="password"/>

At the bottom of the form are two buttons: "Save" and "[Back]".

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process data for the page to edit a board post - 1
  - Write input\_modify.php to process data passed by modify.php
    - Search for the post number's password first, then check for a password match
    - If the password doesn't match, go back to the previous page.
    - If the password matches, use the update statement to edit the post and move back to the post list.

```
$ sudo vi /var/www/html/input_modify.php
<?php
 include "db_conn.php";

 $pw = $_POST['passwd'];
 $idx = $_POST['num'];
 $title = $_POST['title'];
 $content = $_POST['content'];

 $query = "select pw from board where idx=$idx";
 $result = mysqli_query($conn, $query);
 $re_cnt = mysqli_num_rows($result);
 if($re_cnt > 0) {
 $row = mysqli_fetch_array($result);
 }
 :
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process data for the page to edit a board post - 2

```
$ sudo vi /var/www/html/input_modify.php
:
if($pw != $row['pw']) {
 echo "
 <script>
 alert(\"Wrong Password. Try again!\");
 history.go(-1);
 </script>
 ";
}
else {
 $query = "update board set title='$title', content='$content' where idx='$idx'";
 mysqli_query($conn, $query);
 echo "
 <script>
 alert(\"Modify Success!!!!\");
 window.location.href='/list.php';
 </script>
 ";
}
mysqli_free_result($result);
mysqli_close($conn);
?>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process data for the page to edit a board post

Edit Post

192.168.0.142/modify.php?num=1

Title	<input type="text" value="First"/>
Name	acs
E-Mail	acs@acs.com
Date	2024-01-08 17:48:24
Contents	<div style="border: 1px solid black; padding: 10px;">First Value Modify</div>
Password	<input type="password" value="...."/>
<a href="#">Save</a>	
<a href="#">[Back]</a>	

View post

192.168.0.142/view.php?num=1

Title	First
Name	acs
E-Mail	acs@acs.com
Date	2024-01-08 17:48:24
Contents	First Value Modify
<a href="#">[List]</a> <a href="#">[Edit]</a> <a href="#">[Delete]</a>	

# PHP – implementing board page

Lab exercise to implement a simple PHP board

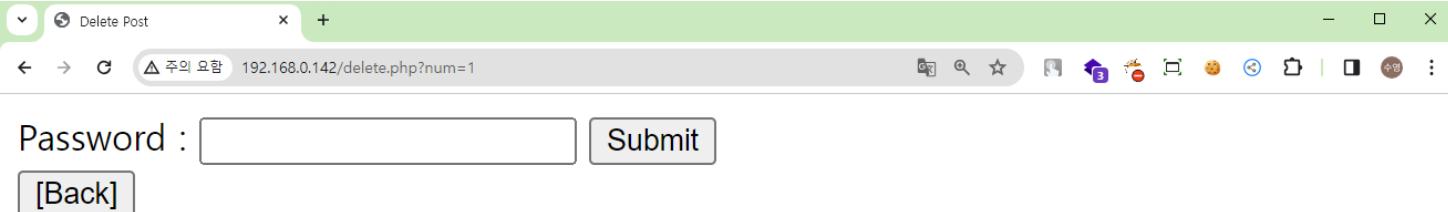
- Implement a page to delete a board post
  - Send the post number from view.php to the hidden attribute of the <input> tag
  - Enter and submit a password to verify the identity of the post you're deleting.
  - Implement a back button to return to the view page so that the deletion can still be undone,
  - Set to send the data to be submitted in the input\_delete.php file.

```
$ sudo vi /var/www/html/delete.php
<!DOCTYPE HTML>
<html>
 <head>
 <meta charset="UTF-8">
 <title>Delete Post</title>
 </head>
 <body>
 <form action="/input_delete.php" method="post">
 Password : <input type="password" name="passwd">
 <input type="hidden" value=<?php echo $_GET['num']; ?> name="num">
 <input type="submit" value="Submit">
 </form>
 <input type="button" value="[Back]" onClick = history.go(-1)>
 </body>
</html>
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Implement a page to delete a board post



A screenshot of a web browser window titled "Delete Post". The address bar shows the URL "192.168.0.142/delete.php?num=1". The page content includes a label "Password : " followed by an input field, a "Submit" button, and a "[Back]" button.

Password :  Submit  
[Back]

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process data for the page to delete a board post - 1
  - Compare the post number and password values sent via delete.php together to check for a match.
  - Proceed with deleting the post if it matches and go to the list, and if it doesn't match go back to the previous page.

```
$ sudo vi /var/www/html/input_delete.php
<?php
 include "db_conn.php";

 $pw = $_POST['passwd'];
 $idx = $_POST['num'];

 $query = "select pw from board where idx=$idx";
 $result = mysqli_query($conn, $query);
 $re_cnt = mysqli_num_rows($result);
 if($re_cnt > 0) {
 $row = mysqli_fetch_array($result);
 if($pw != $row['pw']) {
 echo "
 <script>
 alert(\"Wrong Password. Try again!\");
 history.go(-1);
 </script>
 ";
 }
 }
:
```

# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process data for the page to delete a board post - 2

```
$ sudo vi /var/www/html/input_delete.php
:
else {
 $query = "delete from board where idx='$idx'";
 mysqli_query($conn, $query);
 echo "
 <script>
 alert(\"Delete Success!!!!\");
 window.location.href='/list.php';
 </script>
 ";
}
mysqli_free_result($result);
mysqli_close($conn);
?>
```

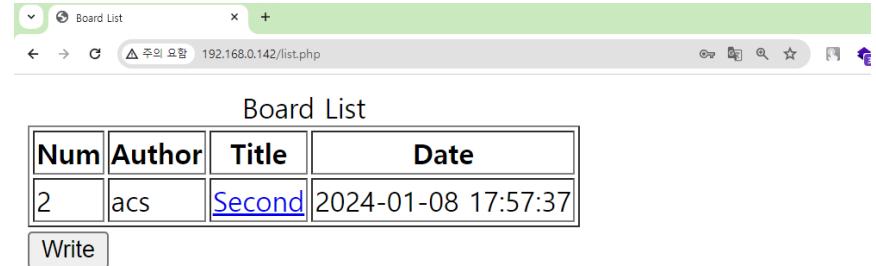
# PHP – implementing board page

Lab exercise to implement a simple PHP board

- Process data for the page to delete a board post



A screenshot of a web browser window titled "Delete Post". The address bar shows the URL "192.168.0.142/delete.php?num=1". The form contains a password input field with the value "...." and a "Submit" button. Below the form is a "[Back]" button.



A screenshot of a web browser window titled "Board List". The address bar shows the URL "192.168.0.142/list.php". The page displays a table titled "Board List" with four columns: Num, Author, Title, and Date. The data row shows: Num 2, Author acs, Title [Second](#), and Date 2024-01-08 17:57:37. A "Write" button is located below the table.

Num	Author	Title	Date
2	acs	<a href="#">Second</a>	2024-01-08 17:57:37