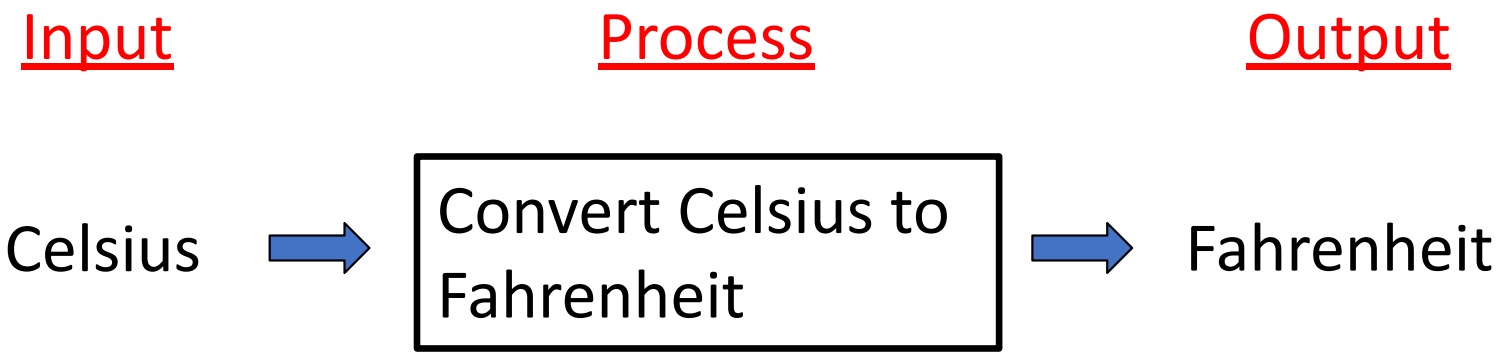
Module 2
Elementary Programming Input, Processing and Output
&
Module 3
Decision Structures and Boolean Logic

Input, Processing, and Output (IPO)

- Typically, computer performs three-step process
  - Receive input
    - Input: any data that the program receives while it is running
  - Perform some process on the input
    - Example: mathematical calculation
  - Produce output

IPO Example: Temperature Converter



- Prompt the user for input (Celsius)
- Process it to convert it to Fahrenheit using F = 9/5(C) + 32
- Output the result by displaying it on the screen.

- Prompt the user for input (Celsius)
- Process it to convert it to Fahrenheit using F = 9/5(C) + 32
- Output the result by displaying it on the screen.

[1]:
```
#
# An IPO Example: Convert Celsius to Fahrenheit
#
```

[2]:
```
celsius = float(input("What is the Celsius temperature?"))
```

```
What is the Celsius temperature? 25.6
```

[3]:
```
fahrenheit = (9 / 5) * celsius + 32
```

[4]:
```
print("The temperature is ", fahrenheit, " degrees Fahrenheit.")
```

```
The temperature is  78.08000000000001  degrees Fahrenheit.
```

Displaying Output with the `print` Function

```
[4]: print("The temperature is ", fahrenheit, " degrees Fahrenheit.")
```

```
The temperature is  78.08000000000001  degrees Fahrenheit.
```

- Function: piece of prewritten code that performs an operation
- `print` function: displays output on the screen
- Argument: data given to a function
  - Example: data that is printed to screen; often involve some **String**s.

Examples

- (Trick) Use double quote to contain single quote and single quote to contain double quote

```
[4]: print("Don't know.")
     print("I'm here!")
```

```
Don't know.
I'm here!
```

```
[5]: print('Einstein said "Everything should be made as simple as possible, but no simpler."')
```

```
Einstein said "Everything should be made as simple as possible, but no simpler."
```

- (Trick) Triple quote (''' or """) can contain single quote and double quote
- (Trick) Triple quote can be used to surround multiline strings.

```
[6]: print("""I'm reading "Hamlet" tonight.""")
```

```
I'm reading "Hamlet" tonight.
```

```
[7]: print("""One
     Two
     Three""")
```

```
One
Two
Three
```

Let us try it and learn more on `print` function! (M2_PrintFunction.ipynb)

Variables

```
[2]: celsius = float(input("What is the Celsius temperature?"))
```

```
What is the Celsius temperature? 25.6
```

```
[3]: fahrenheit = (9 / 5) * celsius + 32
```

- Variable: to access and manipulate data stored in memory
  - A variable references the value it represents
- Assignment statement: to create a variable and make it reference data
  - General format is `variable = expression`
    - Example: `age = 29`
    - Assignment operator: the equal sign (=)

Variable Reassignment

- Variables can reference different values while program is running
- A variable can refer to item of any type
  - Variable that has been assigned to one type can be reassigned to another type

The variable x references an integer


```
1  x = 99
2  print(type(x))
3  print(x)
```

```
<class 'int'>
99
```

The variable x references a string


```
1  x = 'Take me to your leader'
2  print(type(x))
3  print(x)
```

```
<class 'str'>
Take me to your leader
```

Reading Numbers with the `input` Function

- `input` function always returns a string
- Built-in functions convert between data types
  - `int(`*`item`*`)` converts *`item`* to an `int`
  - `float(`*`item`*`)` converts *`item`* to a `float`
  - Nested function call: general format: *`function1`*`(`*`function2`*`(`*`argument`*`))`
    - value returned by function2 is passed to function1

```
[2]: celsius = float(input("What is the Celsius temperature?"))
```

Example

- Write a program to ask the user to enter his/her name, age and income and display these data.

```
[3]: name = input('What is your name? ')
     age = int(input('What is your age? '))
     income = float(input('What is your income? '))
```

```
What is your name?  Chris
What is your age?  20
What is your income?  35000.0
```

```
[4]: print('Here is the data you entered:')
     print('Name:', name)
     print('Age:', age)
     print('Income:', income)
```

```
Here is the data you entered:
Name: Chris
Age: 20
Income: 35000.0
```

Let us try it! (M2_InputLong.ipynb)
What happens if you enter xyz for age?

Performing Calculations

- Math expression: performs calculation and gives a value
  - Math operator: tool for performing calculation
  - Operands: values surrounding operator
    - Variables can be used as operands
- Two types of division:
  - / operator performs floating point division
  - // operator performs integer division
    - Positive results truncated, negative rounded away from zero

Table 2-3 Python math operators

Symbol	Operation
+	Addition
–	Subtraction
*	Multiplication
/	Division
//	Integer division
%	Remainder
**	Exponent

```
[1]: 5 / 2

[1]: 2.5

[2]: 5 // 2

[2]: 2

[3]: -5 // 2

[3]: -3
```

Operator Precedence and Grouping with Parentheses

- Python operator precedence:
  1. Operations enclosed in parentheses
     - Forces operations to be performed before others
  2. Exponentiation (**)
  3. Multiplication (*), division (/ and //), and remainder (%)
  4. Addition (+) and subtraction (-)

precedencec ↓

precedencec →

- Higher precedence performed first
  - Same precedence operators execute from left to right

Example

- Get a number of μm from the user and convert it to cm, mm and μm.

```
[1]: total_microns = float(input('Enter the total number of microns: '))
```

```
Enter the total number of microns:  37659
```

```
[2]: #Get the number of cm
     cm = total_microns // 10000
```

```
[3]: #Get the number of remaining mm
     mm = (total_microns // 1000) % 10
```

```
[4]: #Get the number of remaining microns
     um = total_microns % 1000
```

```
[5]: print('Here is the total', total_microns, 'um in cm, mm, and um:')
     print('Centimeters: ', cm)
     print('Millimeters: ', mm)
     print('Microns: ', um)
```

```
Here is the total 37659.0 um in cm, mm, and um:
Centimeters:  3.0
Millimeters:  7.0
Microns:  659.0
```

Recap and More: Breaking Long Statements into Multiple Lines

- Multiline continuation character (\): Allows to break a long statement into multiple lines

```
result = var1 * 2 + var2 * 3 + \
         var3 * 4 + var4 * 5
```

- **A statement enclosed in parentheses can be broken without \.**

```
print("Monday's sales are", monday,
      "and Tuesday's sales are", tuesday,
      "and Wednesday's sales are", Wednesday)
total = (value1 + value2 +
         value3 + value4 +
         value5 + value6)
```

More About Data Output

- `print` function displays line of output
  - Newline character at end of printed data
  - Special argument `end='delimiter'` causes `print` to place *`delimiter`* at end of data instead of newline character

```
[7]: print('One')
     print('Two')
     print('Three')
```

```
One
Two
Three
```

```
1 print('One', end = '\n')
2 print('Two', end = '\n')
3 print('Three')
```

```
One
Two
Three
```

```
[8]: print('One', end = ' ')
     print('Two', end = ' ')
     print('Three')
```

```
One Two Three
```

```
1 print('One', end = ', ')
2 print('Two', end = ', ')
3 print('Three')
```

```
One, Two, Three
```

Recap and More: More About Data Output (cont'd.)

- Special characters appearing in string literal
  - Preceded by backslash (\)
    - Examples: newline (\n), horizontal tab (\t)
  - Treated as commands embedded in string

Table 2-8 Some of Python's escape characters

Escape Character	Effect
\n	Causes output to be advanced to the next line.
\t	Causes output to skip over to the next horizontal tab position.
\'	Causes a single quote mark to be printed.
\"	Causes a double quote mark to be printed.
\\	Causes a backslash character to be printed.

```
[12]: print("Mon\tTue\tWed")
Mon     Tue     Wed

[13]: print('The path is C:\\temp\\data')
The path is C:\temp\data
```

Formatting Numbers

- Can format display of numbers on screen using built-in `format` function
  - Two arguments:
    - Numeric value to be formatted
    - Format specifier
  - Returns string containing formatted number

```
[14]: print(format(12345.6789, '.2f'))
12345.68

[15]: print(format(12345.6789, '.2e'))
1.23e+04
```

Examples

- Set field width and print numbers aligned in columns.

```
[15]: num1 = 127.899
      num2 = 3456.123
      num3 = 799.999
      num4 = 3.1
```

```
[16]: print(format(num1, '7.2f'))
      print(format(num2, '7.2f'))
      print(format(num3, '7.2f'))
      print(format(num4, '7.2f'))
```

```
 127.90
3456.12
 800.00
   3.10
```

- Insert comma separators

```
[17]: print(format(12345.6712, ',.2f'))
```

```
12,345.67
```

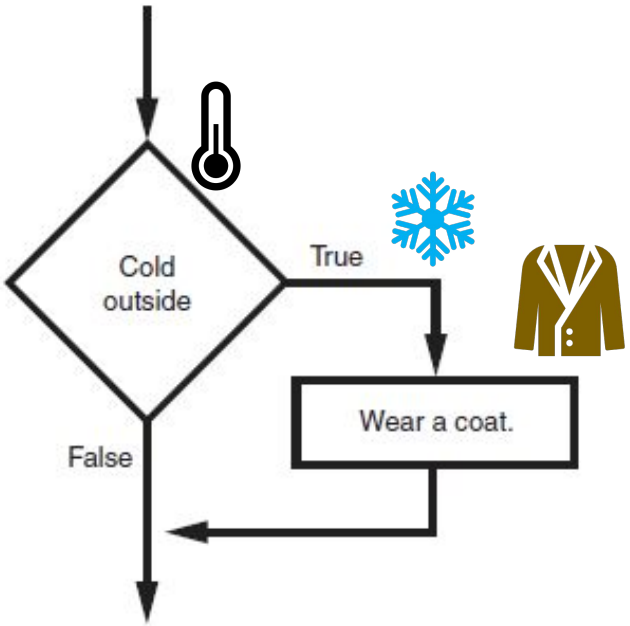
Summary

- This module covered:
  - Input, Processing, and Output
  - Displaying Output with `print` Function
  - Comments
  - Variables
  - Query variable's `type`
  - Reading Input from the Keyboard
  - Performing Calculations
  - More About Data Output
  - Formatting Numbers

Decision Structure

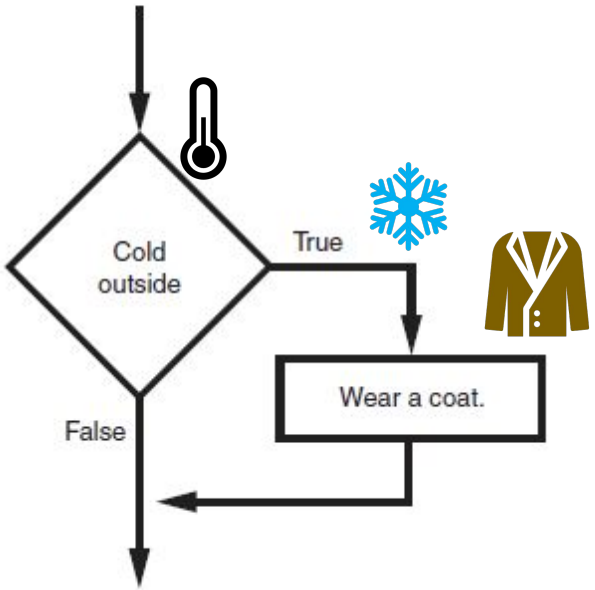
```
if a < b:

elif a == b :

else:
```

The `if` Statement



- **In flowchart, diamond represents true/false condition that must be tested**
- **Actions can be *conditionally executed***
  - Performed only when a condition is true
- **<u>Single alternative decision structure</u>: provides only one alternative path of execution**
  - If condition is not true, exit the structure

The `if` Statement (cont'd.)



- **Python syntax:**

```
if condition:
    Statement
    Statement
```

- **First line known as the `if` clause**
  - Includes the keyword `if` followed by condition
    - The condition can be **true** or **false**
    - When the `if` statement executes, the condition is tested, and if it is true the block statements are executed. Otherwise, block statements are skipped

Boolean Expressions and Relational Operators

Boolean expression: expression tested by if statement to determine if it is true or false

- Example: a > b
- `true` if a is greater than b; `false` otherwise

```
1 20 > 10
```

```
True
```

```
1 10 > 20
```

```
False
```

Relational operator: determines whether a specific relationship exists between two values

- Example: greater than (>)

Exercise (Answer)

- Ask the user to enter three testing scores (0-100). Find the average. Print the average up to two significant digits and congratulate the user if the average is equal or higher than 90.

[1]:
```
test1 = float(input('Enter the first score: '))
test2 = float(input('Enter the second score: '))
test3 = float(input('Enter the third score: '))
```

```
Enter the first score:  90
Enter the second score:  92
Enter the third score:  94.5
```

[2]:
```
average = (test1 + test2 + test3) / 3
```

[3]:
```
print('The average score is', format(average,'.2f'))
```

```
The average score is 92.17
```

[4]:
```
if (average >= 90):
    print('Congratulations!')
    print('This is a great average.')
```

```
Congratulations!
This is a great average.
```

The `if-elif-else` Statement

- `if-elif-else` statement: special version of a decision structure
  - Makes logic of nested decision structures simpler to write
    - Can include multiple `elif` statements
  - Syntax:

```
if condition_1:
    statement(s)
elif condition_2:
    statement(s)
elif condition_3:
    statement(s)
else:
    statement(s)
```

Insert as many `elif` clauses as necessary.

Exercise

- NCKU grading system maps your score in number grade (百分數) to that in letter grade (等第). Write a program that allows the user to input a score and display a letter grade. As an exercise for simplicity, let us treat the score below 77 as an invalid input.

百分數		等第
90-100	95	A+
85-89	87	A
80-84	82	A-
77-79	78	B+
73-76	75	B
70-72	70	B-
67-69	68	C+
63-66	65	C
60-62	60	C-
≦ 59	50	F
0	0	X

Program Output (with input shown in **bold**)

Enter a score: **83.2**

The score 83.2 is A-

Program Output (with input shown in **bold**)

Enter a score: **75**

The score 75.0 is an invalid input

Exercise (Answer)

- NCKU grading system maps your score in number grade (百分數) to that in letter grade (等第). Write a program that allows the user to input a score and display a letter grade. As an exercise for simplicity, let us treat the score below 77 as an invalid input.

```
[ ]: score = float(input('Enter a score:'))
```

```
[ ]: if (score >= 90):
         grade = 'A+'
     elif (score >= 85):
         grade = 'A'
     elif (score >= 80):
         grade = 'A-'
     elif (score >= 77):
         grade = 'B+'
     else:
         grade = 'an invalid input'
```

```
[ ]: print('The score', score, 'is', grade)
```

百分數		等第
90-100	95	A+
85-89	87	A
80-84	82	A-
77-79	78	B+
73-76	75	B
70-72	70	B-
67-69	68	C+
63-66	65	C
60-62	60	C-
≦ 59	50	F
0	0	X

Logical Operators

Logical operators: operators that can be used to create complex Boolean expressions

- `and` operator and `or` operator: binary operators, connect two Boolean expressions into a compound Boolean expression
- `not` operator: unary operator, reverses the truth of its Boolean operand

Table 3-4 Compound Boolean expressions using logical operators

Expression	Meaning
`x > y and a < b`	Is x greater than y AND is a less than b?
`x == y or x == z`	Is x equal to y OR is x equal to z?
`not (x > y)`	Is the expression x > y NOT true?

Example

- A bank will grant the loan if one earns 500,000 per year and is employed for 2 years. Write a program to determine whether the user qualifies for a loan.

```
[ ]: MIN_SALARY = 500000
     MIN_YEARS = 2

[ ]: salary = float(input('Enter your annual salary:'))

[ ]: years_on_job = float(input('Enter the number of years employed:'))

[ ]: if salary > MIN_SALARY and years_on_job > MIN_YEARS:
         print('You quality for the loan.')
     else:
         print('You do not quality for the loan.')
```

Summary

- This module covered:
  - Decision structures, including:
    - Single alternative decision structures
    - Dual alternative decision structures
    - Nested decision structures
  - Relational operators and logical operators as used in creating Boolean expressions
  - String comparison as used in creating Boolean expressions
  - Boolean variables