

M110: Python Programming

Meeting #2

Fundamentals of Python Programming



Content

- Introduction to Python
- Running Python Programs
- Data Types and Variables
- Using Numeric and String Variables
- Printing with Parameters
- Getting Input from a User

Why Python?

Python is object-oriented

- Supports concepts such as polymorphism, operation overloading, and multiple inheritance

It's free (open source)

- Downloading and installing Python is free and easy
- Source code is easily accessible
- Free doesn't mean unsupported! Online Python community is huge

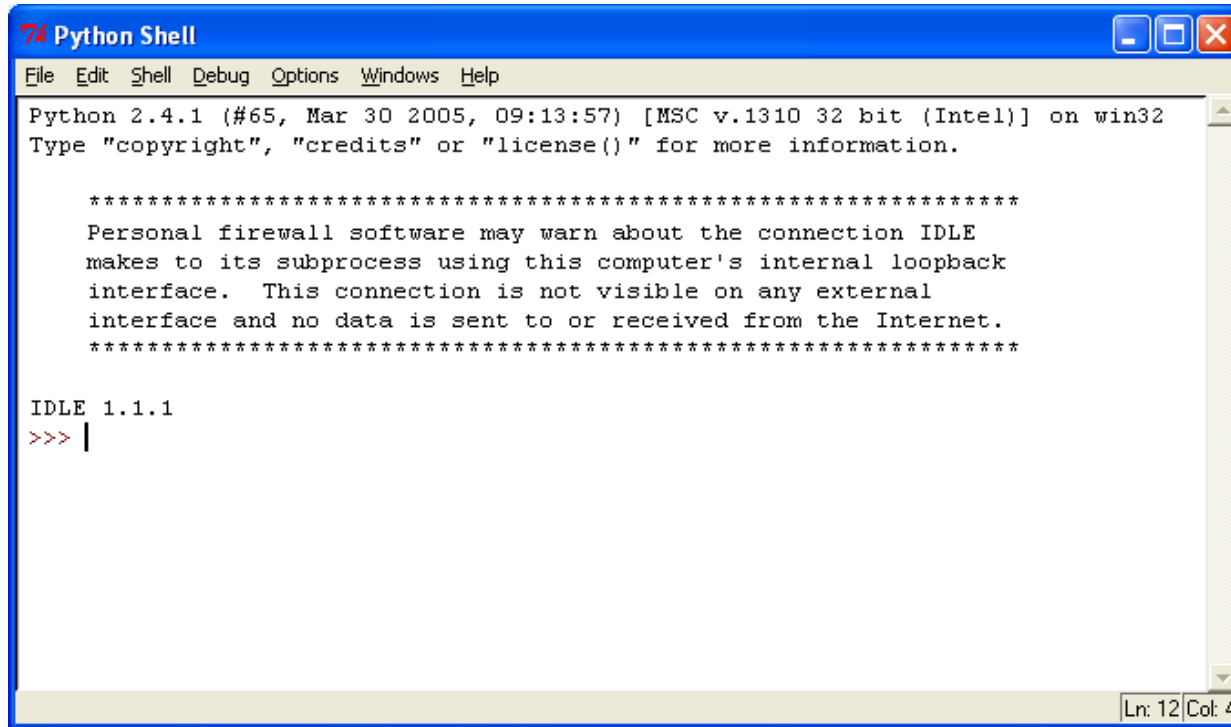
It's portable

- Python runs virtually on major platforms used today
- As long as you have a compatible Python interpreter installed, Python programs will run in exactly the same manner, irrespective of platform

It's powerful

- Dynamic typing
- Built-in types and tools
- Library utilities
- Third party utilities (e.g. Numeric, NumPy, SciPy)
- Automatic memory management

Python IDLE

The image shows a screenshot of the Python IDLE Python Shell window. The window has a blue title bar with the text "Python Shell" and standard window control buttons (minimize, maximize, close). Below the title bar is a menu bar with the following options: File, Edit, Shell, Debug, Options, Windows, and Help. The main text area contains the following text:

```
Python 2.4.1 (#65, Mar 30 2005, 09:13:57) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface.  This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.1.1
>>> |
```

The status bar at the bottom right of the window shows "Ln: 12 | Col: 4".

- **IDLE: Integrated DeveLopment Environment**
- After installing the IDLE, you can start writing your Python programs.

Programming Modes in Python

- **Interactive Mode**

- gives you immediate feedback
- Not designed to create programs to be saved and run later

- **Script Mode**

- Write, edit, save, and run (later)
- Save your file using the “.py” extension

Create and run programs in Script Mode

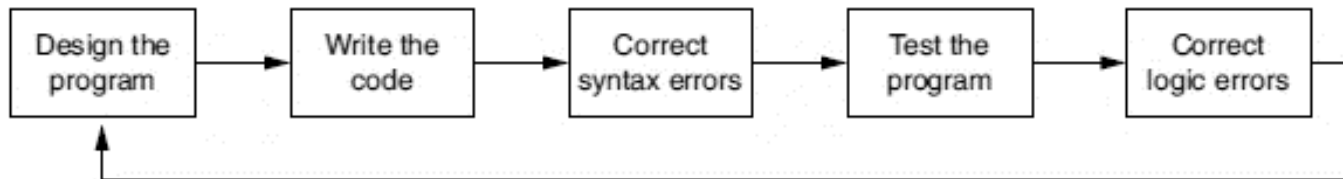
1. Go to the File menu.
2. Make a new file.
3. Give a name for the new file such as:
firstProgram.py and then save with **.py** extension.
4. You can now start writing your code.
5. To run your code, save it first and then go to the run menu →
choose run Module or press F5.

The Program Development Cycle

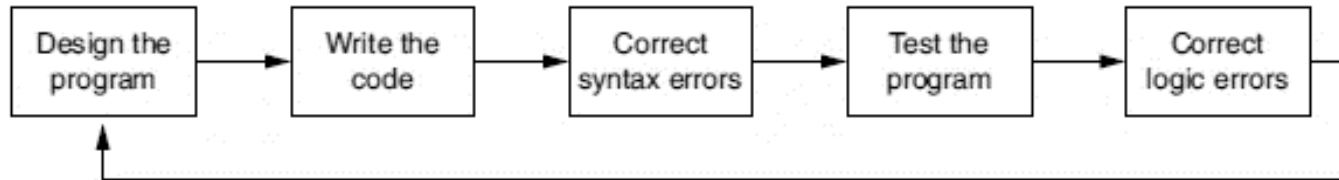
Programs must be carefully designed before they are written. During the design process, programmers use tools such as pseudocode and flowcharts to create models of programs (as we have seen in lecture one).

The process of creating a program that works correctly typically requires the five phases shown in the below Figure.

The entire process is known as the **program development cycle**.



The Program Development Cycle



1. **Design the Program:** A program should be carefully designed before the code is written.
2. **Write the Code:** After designing the program, the programmer begins writing code in a high-level language such as Python, considering the proper syntax.
3. **Correct Syntax Errors:** If the program contains a syntax error, the compiler or interpreter will display an error message indicating what the error is.
4. **Test the Program:** Once the code is in an executable form, it is then tested to determine whether any **logic errors** exist.
A **logic error** is a mistake that does not prevent the program from running but causes it to produce incorrect results.
5. **Correct Logic Errors:** If the program produces incorrect results, the programmer debugs the code.

Data Types in Python

Python supports different Data types:

- **integer (signed integers):** positive or negative whole numbers with no decimal point.
- **float (floating point real values):** real numbers and are written with a decimal point dividing the integer and fractional parts. Floats may also be in scientific notation, with E or e indicating the power of 10 ($2.5e2 = 2.5 \times 10^2 = 250$).
- **complex (complex numbers):** are of the form $a + bJ$, where a and b are floats and J (or j) represents the square root of -1 (which is an imaginary number). The real part of the number is a , and the imaginary part is b . Complex numbers are not used much in Python programming.

Data Types in Python

Python supports different Data types (cont'd):

- **Boolean:** The Boolean data type is represented in Python as type **bool**. It has one of the two values **True** or **False**. This type is used to compare two values.

>>5==4 ➡ False

>>5>=4 ➡ True

- **String:** A string in Python can be created using single, double, and triple quotes.
- **Sequence Types:** list, tuple, range (to be discussed later)

N.B: The two consecutive equal marks operator (==) returns true if both operands have the same value; otherwise, it returns false .

Python print() Function

A **function** is a piece of prewritten code that performs an operation. Python has numerous built-in functions that perform various operations. One of the most fundamental built-in functions is the print function.

The **print()** function prints the specified message to the screen, or other standard output device.

The message can be a string, or any other object, the object will be converted into a string before written to the screen.

```
print("Hello!", "How are you?")
```

Hello! How are you?

```
x = ("apple", "banana", "cherry")  
print(x)
```

('apple', 'banana', 'cherry')

These pieces of data are sequences of characters. In programming terms, a sequence of characters that is used as data is called a **string**. When a string appears in the actual code of a program, it is called a **string literal**. In Python code, string literals must be enclosed in quote marks (e.g: **"apple"**) .

Literals are numbers or strings that appear directly in a program.

Your First Python Program

- Python is "case-sensitive":

- `print("hello")` #correct
- `print('hello')` #correct
- `Print("hello")` #error
- `PRINT("hello")` #error

- "hello" is a **String** literal.
- When the computer does not recognize the statement to be executed, a **syntax error** is generated.

String Literals

- String literals in python are **immutable**, which means once they are created, they cannot be changed. Strings are surrounded by either single quotation marks, or double quotation marks.
 - 'hello' is the same as "hello".
- Strings can be output to screen using the **print** function. For example: `print("hello")`.
- If you want a string literal to contain either a single-quote or an apostrophe as part of the string, you can enclose the string literal in double-quote marks.

```
print("Don't panic!")
```



Don't panic!

- Likewise, you can use single-quote marks to enclose a string literal that contains double quotes as part of the string.

```
print('Your TMA is about " ICT Tech" ')
```



Your TMA is about "ICT Tech"

String Literals

- Strings in Python are arrays of bytes representing Unicode characters.
 - Python does not have a character data type.
 - A single character is simply a string with a length of 1.
- Square brackets can be used to access elements of the string.

Example:

Get the first character of a string (In Python, the 1st character has position 0):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	Index/position
H	e	l	l	o	,		A	O	U		s	t	u	d	e	n	t	s	!	

```
s = "Hello, AOU students!"  
print(s[1])
```



- To get the length of a string, use the **len()** function, which returns the length of the string.

```
s = "Hello, AOU students!"  
print(len(s))
```



String Literals

- To check if a certain phrase or character is present in a string, we can use the keyword **in**.

```
s = "Hello, AOU students!"  
print('AOU' in s)
```



True

- You can return a range of characters using **Slicing**.
- Slicing format: *string[start: stop: step]***
 - It will return a string containing a copy of the characters from **start** up to, but not including, **stop**. The **step** identifies how steps should be taken from **start** to **stop**(exclusive).

String Literals

Index/position

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
H	e	l	l	o	,		A	O	U		s	t	u	d	e	n	t	s	!

Examples:

- Get the characters from position 1 to position 4:

```
s = "Hello, AOU students!"  
print(s[1:5])
```

→ ello

- You can specify the steps:

```
s = "Hello, AOU students!"  
print(s[0:5:2])
```

→ Hlo

- By leaving out the **start** index, the range will start at the first character.

```
s = "Hello, AOU students!"  
print(s[:5])
```

→ Hello

- By leaving out the **stop** index, the range will go to the end.

```
s = "Hello, AOU students!"  
print(s[7:])
```

→ AOU students!

String Literals

- You can use negative indexes to start the slice from the end of the string.

```
b = "Students!"  
print(b[-6:-1])
```



dents

```
b = "Students!"  
print(b[-6:-1:2])
```

Using step value 2



dns

Positive indexing

0	1	2	3	4	5	6	7	8
S	t	u	d	e	n	t	s	!
-9	-8	-7	-6	-5	-4	-3	-2	-1

Negative indexing

More about the Print function

You will learn now more details about the Python print function, and you'll see techniques for formatting output in specific ways.

Overpowering the print Function's Ending Newline

- By default, the value of this parameter is '\n' (the new line character).
- Python's print() function comes with a parameter called 'end'.
- You can end a print statement with any character or string using this parameter.

```
print('I', end="")  
print('Love', end="")  
print('Python')
```



ILovePython

Notice in the argument end="" there is no space between the quote marks. This specifies that the print function should print nothing at the end of its output. However, you can include whatever you want between the quotes if you want so.

```
print('I', end='*')  
print('Love', end=' ')  
print('Python')
```



I*Love Python

More about the Print function

Specifying an Item Separator

When multiple arguments are passed to the print function, they are automatically separated by a space when they are displayed on the screen.

`print('I', 'Love', 'Python')` ➡ **I Love Python**

If you do not want a space printed between the items (or want anything else to be inserted, you can pass the argument **sep=""** to the print function, as shown below:

`print('I', 'Love', 'Python', sep="")` ➡ **ILovePython**

`print('I', 'Love', 'Python', sep='*')` ➡ **I*Love*Python**

More about the Print function

Escape Characters

An escape character is a special character that is preceded with a backslash (`\`), appearing inside a string literal. When a string literal that contains escape characters is printed, the escape characters are treated as special commands that are embedded in the string. For example, `\n` is the newline escape character. Several escape characters are recognized by Python, some of which are listed below:

- `\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.

```
print('I\nLove\nPython')
```



```
I
Love
Python
```

```
print('I\tLove\tPython')
```



```
I      Love      Python
```

More about the Print function

Displaying Multiple Items with the + Operator

When the + operator is used with two strings, however, it performs string concatenation.

`print('This is ' + 'one string.')` ➡ This is one string.

Formatting Numbers

When a floating-point number is displayed by the print function, it can appear with up to 12 significant digits. When you call the built-in *format function*, you pass two arguments to the function: a **numeric value** and a *format specifier*. The *format specifier* is a string that contains special characters specifying how the numeric value should be formatted.

`print(format(12345.6789, '.2f'))` ➡ 12345.68

Program Documentation

- Comments are short notes placed in different parts of a program, explaining how those parts of the program work.
- Comment lines provide documentation about your program.
 - Anything after the “#” symbol is a comment
 - Ignored by the computer


```
# First Python Program  
# January 30, 2025  
# print("Hello!")
```

Variables

- A **variable** is a name that represents a value stored in the computer's memory.
- Variables let us store and reuse values in several places.
- Programs use variables to access and manipulate data that is stored in memory.
- To do this we need to define the variable and then tell it to refer to a value.
- We do this using an assignment statement. ***variable = expression***

Example:

```
y = 3  
print(y)
```



3

Variables

- You can also assign to multiple names at the same time.

- Example1:

```
>>> x,y = 2,3
```

```
>>> x
```

2

```
>>> y
```

3

- Example2:

```
>>> x = 5; y = 4;
```


Variables– Rules

- Variable names can contain letters, numbers, and the underscore (the dollar sign is NOT accepted!).
- Variable names cannot contain spaces.
- Variable names cannot start with a number.
- Variable name cannot be a reserved word.
- Case matters: temp and Temp are different variables.
- There are many reserved words such as:
and, not, or, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, pass, print, raise, return, try, while

WARNING! You cannot use a variable until you have assigned a value to it. An error will occur if you try to perform an operation on a variable, such as printing it, before it has been assigned a value.

Displaying Multiple Items with the print Function

Python allows us to display multiple items with one call to the print function.

We simply need to separate the items with commas as shown in the following program:

```
# Displaying multiple items with one call
Course_Name="Python Programming"
Course_Section=210
print("The course is",Course_Name,", and my section is",Course_Section )
```

Output:

The course is Python Programming , and my section is 210

Reading from the keyboard

- To read from the keyboard, you normally use the **input** function in an assignment statement that follows this general format:

variable = input(prompt)

```
x=input("enter your text: ")#Hello Ahmad  
print(x)
```



Enter your text: Hello Ahmad
Hello Ahmad

- For reading values (numbers) from the keyboard we can use “**eval()**” which converts the string to values.

```
x=eval(input("enter a number: "))  
y=eval(input("enter another number: "))  
print("The sum of both numbers is:",x+y)
```



enter a number: 5
enter another number: 10
The sum of both numbers is: 15

Math Operators

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Float Division	1 / 2	0.5
//	Integer Division	1 // 2	0
**	Exponentiation	4 ** 0.5	2.0
%	Remainder	20 % 3	2

Can be used also for string concatenation:
`y="hello"`
`print(y+" world!")`
`==> hello world!`

Can be used also for string repetition:
`print("Hi " * 3)`
`==> Hi Hi Hi`

Operator Precedence

- You can write statements that use complex mathematical expressions involving several operators.
- First, operations that are enclosed in parentheses are performed first. Then, when two operators share an operand, the operator with the higher *precedence* is applied first.
- Parts of a mathematical expression may be grouped with parentheses to force some operations to be performed before others.
- The precedence of the math operators, from highest to lowest, are:
 - a. Parentheses ()
 - b. Exponentiation: **
 - c. Multiplication, division, and remainder: * / // %
 - d. Addition and subtraction: + -

$$5 + 2 * 4 \quad //13$$

$$10 / 2 - 3 \quad //2.0$$

$$8 + 12 * 2 - 4 \quad //28$$

$$6 - 3 * 2 + 7 - 1 \quad //6$$

$$(6 - 3) * (2 + 7) / 3 \quad //9.0$$

Casting in Python

- Python converts numbers internally in an expression containing mixed types to a common type for evaluation.
- Sometimes, you need to explicitly convert a number from one type to another. This is called casting.
 - **int(x)** to convert x to a plain integer.
 - **float(x)** to convert x to a floating-point number.
 - **str()** to construct string from a wide variety of data types, including strings, integer literals and float literals
 - **complex(x)** to convert x to a complex number with real part x and imaginary part zero.
 - **complex(x,y)** to convert x and y to a complex number with real part x and imaginary part y. x and y are numeric expressions.

Casting in Python

```
>>> x = '100'
```

```
>>> y = '-90'
```

```
>>> print (x + y)
```

```
100-90
```

Since they are strings, x and y will be concatenated

```
>>> print (int(x) + int(y))
```

```
10
```

Since they have been casted, values of x and y will be added

Casting in Python

- Casting to integers:

```
x=int(input("enter the value")) #5
```

```
y=int(input("enter the value")) #10
```

➡ $x+y = 15$

- Casting to floats:

```
x=float(input("enter the value")) #5.0
```

```
y=float(input("enter the value")) #10.0
```

➡ $x+y= 15.0$

- Casting to strings:

```
x = str("s1") # x will be 's1'
```

```
y = str(2) # y will be '2'
```

```
z = str(3.0) # z will be '3.0'
```


More about Strings

String Testing Methods

Method	Description
isalnum()	Returns true if the string contains only alphabetic letters or digits and is at least one character in length. Returns false otherwise.
isalpha()	Returns true if the string contains only alphabetic letters and is at least one character in length. Returns false otherwise.
isdigit()	Returns true if the string contains only numeric digits and is at least one character in length. Returns false otherwise.
islower()	Returns true if all of the alphabetic letters in the string are lowercase, and the string contains at least one alphabetic letter. Returns false otherwise.
isspace()	Returns true if the string contains only whitespace characters and is at least one character in length. Returns false otherwise. (Whitespace characters are spaces, newlines (\n), and tabs (\t).
isupper()	Returns true if all the alphabetic letters in the string are uppercase, and the string contains at least one alphabetic letter. Returns false otherwise.

More about Strings

Modification Methods

Although strings are immutable, meaning they cannot be modified, they do have several methods that return modified versions of themselves.

Method	Description
lower()	Returns a copy of the string with all alphabetic letters converted to lowercase. Any character that is already lowercase, or is not an alphabetic letter, is unchanged.
lstrip()	Returns a copy of the string with all leading whitespace characters removed. Leading whitespace characters are spaces, newlines (\n), and tabs (\t) that appear at the beginning of the string.
lstrip(char)	The char argument is a string containing a character. Returns a copy of the string with all instances of char that appear at the beginning of the string removed.
rstrip()	Returns a copy of the string with all trailing whitespace characters removed. Trailing whitespace characters are spaces, newlines (\n), and tabs (\t) that appear at the end of the string.
rstrip(char)	The char argument is a string containing a character. The method returns a copy of the string with all instances of char that appear at the end of the string removed.
strip()	Returns a copy of the string with all leading and trailing whitespace characters removed.
strip(char)	Returns a copy of the string with all instances of char that appear at the beginning and the end of the string removed.
upper()	Returns a copy of the string with all alphabetic letters converted to uppercase. Any character that is already uppercase, or is not an alphabetic letter, is unchanged.

Drill Examples

```
In [1]: 1 8/3 # classic division always returns a floating point number
```

```
Out[1]: 2.6666666666666665
```

```
In [8]: 1 round(8/3,2) # returns a floating point number with the specified number of decimals.
```

```
Out[8]: 2.67
```

```
In [2]: 1 8//3 # floor division discards the fractional part
```

```
Out[2]: 2
```

```
In [3]: 1 10%3 # the % operator returns the remainder of the division
```

```
Out[3]: 1
```

```
In [4]: 1 2**3 # Exponent
```

```
Out[4]: 8
```

Drill Examples

```
In [5]: 1 text1='doesn\'t' # use \' to escape the single quote
        2 print(text1)
        3 text2="doesn't"
        4 print(text2)
```

doesn't
doesn't

```
In [6]: 1 #Strings can be concatenated (glued together) with the + operator, and repeated with *:
        2 2*"la "+"land"
```

Out[6]: 'la la land'

```
In [7]: 1 #Two or more string literals next to each other are automatically concatenated.
        2 "M" "110" " Course"
```

Out[7]: 'M110 Course'

```
In [9]: 1 #Strings can be indexed, with the first character having index 0
        2 Name="Samira"
        3 Name[4]
```

Out[9]: 'r'

```
In [10]: 1 # Indices may also be negative numbers, to start counting from the right:
        2 Name[-1]
```

Out[10]: 'a'

Drill Examples

```
In [11]: 1 #Slicing is also supported. While indexing is used to obtain individual characters,
          2 # slicing allows you to obtain substring:
          3 Name="Samira"
          4 Name[0:3] # characters from position 0 (included) to 3 (excluded)
```

Out[11]: 'Sam'

```
In [12]: 1 print(Name[:3]) # character from the beginning to position 3 (excluded)
          2 print(Name[3:]) # characters from position 3 (included) to the end
          3 print(Name[-2:]) # characters from the second-last (included) to the end
```

Sam
ira
ra

```
In [13]: 1 # Python strings cannot be changed – they are immutable.
          2 Name[0]="K" # assigning to an indexed position in the string results in an error:
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-13-0febc674b66a> in <module>()
      1 # Python strings cannot be changed – they are immutable.
----> 2 Name[0]="K" # assigning to an indexed position in the string results in an error:

TypeError: 'str' object does not support item assignment
```

```
In [14]: 1 # The built-in function len() returns the length of a string:
          2 word="supermirafiori"
          3 len(word)
```

Out[14]: 14