Fundamentals of Programming

Unit 2
Python Basics

CUSCS

Python Basics

Store our values in our programs!

Quotation

- Quotation
  - accept single (‘), double (") and triple (''' or """") quotes to denote string literals
  - as long as the same type of quote starts and ends the string
  - triple quotes are used to span the string across multiple lines

- Example
  - single (‘) → ‘Hello World’
  - double (") → “Hello World”
  - triple (''' or """") → ‘’’Hello World’’’

Quotation

- Example
  - Input the following code and run the file

Programming Code

```
print('Chan Tai Man')
print("Chan Tai Man")
print('''Chan Tai Man''')
print"""Chan Tai Man""")
```

Example.py

File Edit Format Run Options Window Help

```
print('Chan Tai Man')
print("Chan Tai Man")
print('''Chan Tai Man''')
print"""Chan Tai Man""")
```

Lines and Indentation

- Lines and Indentation
  - Python provides no braces to indicate blocks of code for class and function definitions or flow control
  - blocks of code are denoted by line indentation, which is rigidly enforced

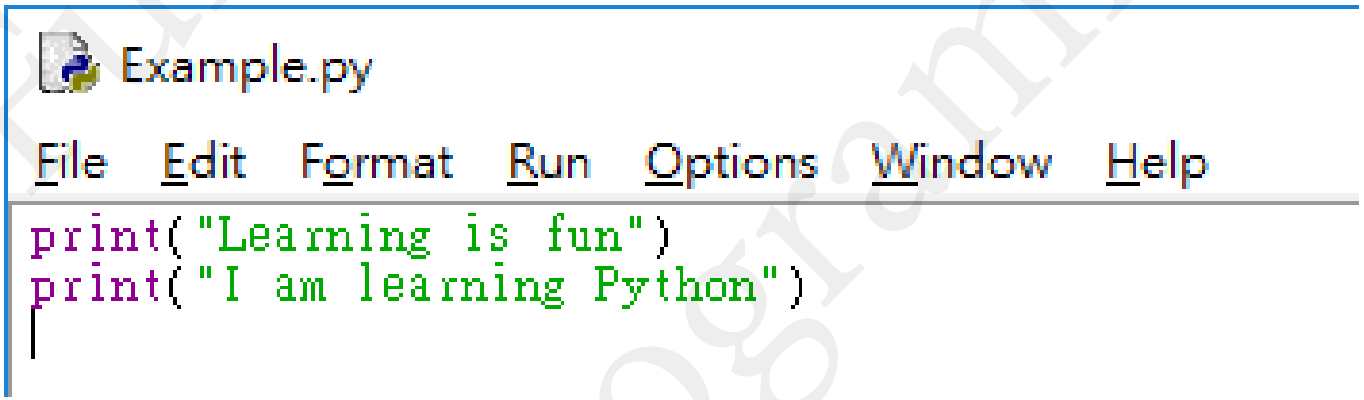
- Features
  - the number of spaces in the indentation is variable, but all statements within the block must be indented the same amount
  - all the continuous lines indented with same number of spaces would form a block

Lines and Indentation

- Example
  - Input the following code and see the results

Programming Code

```
print("Learning is fun")
print("I am learning Python")
```



Lines and Indentation

- Example
  - Input the following code and see the results

Programming Code

```
print("Learning is fun")
    print("I am learning Python")
```

Example.py

File Edit Format Run Options Window Help

```
print("Learning is fun")
    print("I am learning Python")
```

Multi-Line Statements

- Multi-Line Statements
  - statements in Python typically end with a new line
  - Python allows the use of the line continuation character (\) to denote that the line should continue

- Example
  - Input the following code and see the results

Example.py

File Edit Format Run Options Window Help

```
a = 10
b = 20
c = 30
d = a + \
    b + \
    c
print(d)
```

Programming Code

```
a = 10
b = 20
c = 30
d = a + \
    b + \
    c
print(d)
```

Identifiers

Identifiers

a name used to identify a variable, function, class, module or other object

- Features
  - starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9)
  - do not allow punctuation characters such as @, $, and % within identifiers
  - Python is a case sensitive programming language → ***learning*** and ***Learning*** are different identifiers

Identifiers

- Naming Conventions for Identifiers
  - class names start with an uppercase letter
  - all other identifiers start with a lowercase letter
  - starting an identifier with a single leading underscore indicates that the identifier is private → ***_learning***
  - starting an identifier with two leading underscores indicates a strongly private identifier → ***__learning***
  - if the identifier also ends with two trailing underscores, the identifier is a language-defined special name → ***learning__***

Identifiers

- Class Activity
  - Which of the following valid identifiers?

1. abc123
2. @_happy
3. learning_python
4. 5class
5. _5678

Reserved Words

- Reserved Words
  - reserved words that we cannot use them as constant or variable or any other identifier names
  - all the Python keywords contain lowercase letters only

- Examples
  - and
  - not
  - finally
  - assert

Comments

Comments

a hash sign (#) that is not inside a string literal begins a comment
all characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them

- Types of Comments
  - Single-Line Comment
    - start with the hash # character followed by text for further explanation
  - Multi-Line Comments
    - don't have explicit support for multi-line comments but we can use hash # character to the multiple lines

Comments

- Example
  - Input the following code and run the file

Programming Code

```
#display information
#Learning Comments
print("Hello! I am learning Comments") #Comments Here

'''
print("Multiple Line Comments")
This is also a comment.
'''
```

Example.py

File Edit Format Run Options Window Help

```
#display information
#Learning Comments
print("Hello! I am learning Comments") #Comments Here

'''
print("Multiple Line Comments")
This is also a comment.
'''
```

Variables

Variables

a name that is used to refer to memory location

- Features
  - don't need to specify the type of variable → Python is a infer language and smart enough to get variable type
  - variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore
  - variables are a symbolic name that is a reference or pointer to an object
  - variables are used to denote objects by that name

Variables

- Declare Variable and Assignment Variable
  - Python variables do not need explicit declaration to reserve memory space
  - declaration happens automatically when we assign a value to a variable
  - equal sign (=) is used to assign values to variables

- = Operator
  - operand to the left of the = operator → the name of the variable
  - operand to the right of the = operator → the value stored in the variable

Variables

- Example
  - Input the following code and run the file

Programming Code

```
numberOfProduct = 200 #assign an integer
price = 10.5 #Assign a float
name = "Milk" # Assign a string

print(numberOfProduct)
print(price)
print(name)
```

Example.py

File Edit Format Run Options Window Help

```
numberOfProduct = 200 #assign an integer
price = 10.5 #Assign a float
name = "Milk" # Assign a string

print(numberOfProduct)
print(price)
print(name)
```

Variables

- Multiple Assignment
  - Python allows us to assign a single value to several variables simultaneously

- Example
  - Input the following code and run the file

Programming Code

```
a = b = c = 10

print(a)
print(b)
print(c)
```

Example.py

File Edit Format Run Options Window Help

```
a = b = c = 10

print(a)
print(b)
print(c)
```

- an integer object is created with the value 10, and all 3 variables are assigned to the same memory location

Variables

- Example
  - Input the following code and run the file
  - we can also assign multiple objects to multiple variables

Programming Code

```
a, b, c = 10, 20, "Python"

print(a)
print(b)
print(c)
```

Example.py

File Edit Format Run Options Window Help

```
a, b, c = 10, 20, "Python"

print(a)
print(b)
print(c)
```

- two integer objects with values 10 and 20 are assigned to variables ***a*** and ***b*** respectively, and one string object with the value ”learning" is assigned to the variable ***c***

Variables

- Delete a Variable
  - delete the variable using the ***del*** keyword

- Example
  - Input the following code and run the file

Programming Code

```
a = 10
print(a)

del a
print(a)
```

Example.py

File Edit Format Run Options Window Help

```
a = 10
print(a)

del a
print(a)
```

Variables

- Print Single and Multiple Variables
  - print multiple variables within the single print statement

- Example
  - Input the following code and run the file

Programming Code

```
a = 10
print(a)
print((a))
```

Example.py

File Edit Format Run Options Window Help

```
a = 10
print(a)
print((a))
```

- Print single variable

Variables

- Example
  - Input the following code and run the file

Programming Code

```
a = 10
b = 20
print(a, b)

print(10, 20, 30)
```

- Print multiple variables

Example.py

File Edit Format Run Options Window Help

```
a = 10
b = 20
print(a, b)

print(10, 20, 30)
```

Advanced Variable Naming

- Camel Case
  - each word or abbreviation in the middle of begins with a capital letter
  - no intervention of whitespace
  - Example: learningPythonProgramming, valueOfCourse
- Pascal Case
  - the same as the Camel Case, but here the first word is also capital
  - Example: LearningPythonProgramming, ValueOfCourse
- Snake Case
  - Words are separated by the underscore
  - Example: learning_python_programming, value_of_course

Data Types

What is the type of the value?

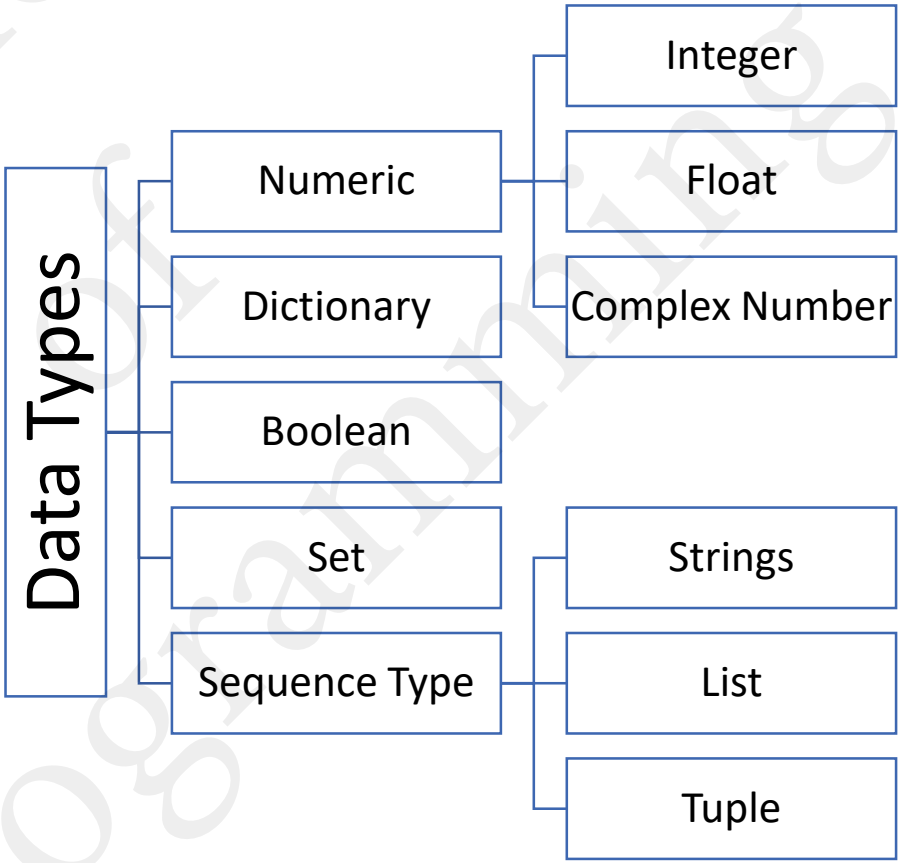
Data Types

- Data Types
  - variables can hold values, and every value has a data-type
  - Python is a dynamically typed language → do not need to define the type of the variable while declaring it
  - the interpreter implicitly binds the value with its type

- Check the Type
  - check the type of the variable used in the program
  - the type() function → returns the type of the variable passed

Standard Data Types

- Standard Data Types
  - a variable can hold different types of values
  - Python provides various standard data types that define the storage method on each of them



Numbers

- Numbers
  - Number stores numeric values
  - the integer, float, and complex values belong to a Python Numbers data-type
  - the ***type()*** function to know the data-type of the variable
  - the ***isinstance()*** function is used to check an object belongs to a particular class

Numbers

- Int
  - integer value can be any length
  - has no restriction on the length of an integer
- Float
  - store floating-point numbers
  - accurate up to 15 decimal points
- Complex
  - a complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively

Numbers

- Example
  - Input the following code and run the file

Programming Code

```
a = 10
print("Type of a = ", type(a))

b = 25.5
print("Type of b = ", type(b))

c = 4+5j
print("Type of c = ", type(c))
print("C is a Complex number? ", isinstance(4+5j, complex))
```

Numbers

- Example
  - Input the following code and run the file

Example.py

File Edit Format Run Options Window Help

```
a = 10
print("Type of a = ", type(a))

b = 25.5
print("Type of b = ", type(b))

c = 4+5j
print("Type of c = ", type(c))
print("C is a Complex number? ", isinstance(4+5j, complex))
```

Numbers

- Class Activity
  - Assign the value 100.5 to the variable called ***price***
  - Display the variable and data type of the variable
  - Check whether price is a complex number and show the result

Sequence Type

- String
  - string can be defined as the sequence of characters represented in the quotation marks
  - we can use single, double, or triple quotes to define a string
  - string handling in Python is a straightforward task → Python provides built-in functions and operators to perform operations in the string
- Features
  - subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end
  - plus (+) sign is the string concatenation operator
  - asterisk (*) is the repetition operator

String

- Example
  - Input the following code and run the file

Programming Code

```
words = "Python Learning"

print(words) #print string
print(words[0]) #print first character of the string
print(words[7:12]) #print characters starting from 7th to 12th
print(words[7:]) #print starting from 7th characters
print(words*2) #print string twice
print(words + " Happy") #Print concatenated string
```

String

- Example
  - Input the following code and run the file

Example.py

File Edit Format Run Options Window Help

```
words = "Python Learning"

print(words) #print string
print(words[0]) #print first character of the string
print(words[7:12]) #print characters starting from 7th to 11th
print(words[7:]) #print starting from 7th characters
print(words*2) #print string twice
print(words + " Happy") #Print concatenated string
```

String

- Class Activity
  - Display the following results with the string “I love eating ice-cream”

```
= RESTART:
I love eating ice-cream
I
ice
ice-cream
I love eating ice-creamI love eating ice-creamI love eating ice-cream
I love eating chooclate ice-cream
```

Sequence Type

- Lists
  - the most versatile of Python's compound data types
  - A list contains items separated by commas and enclosed within square brackets ([])
  - lists are similar to arrays in C → one difference between them is that all the items belonging to a list can be of different data type
- Features
  - values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1
  - plus (+) sign is the list concatenation operator
  - asterisk (*) is the repetition operator

List

- Example
  - Input the following code and run the file

Programming Code

```
myList = [ 'Python', 123 , 4.5, 'Learning', 6.78 ]
mySubList = [901, 'Program']

print (myList)          # print complete list
print (myList[0])       # print first element of the list
print (myList[1:3])     # print elements starting from 2nd till 3rd
print (myList[2:])      # print elements starting from 3rd element
print (mySubList * 2)  # print list twice
print (myList + mySubList) # print concatenated lists
```

List

- Example
  - Input the following code and run the file

Example.py

File Edit Format Run Options Window Help

```
myList = [ 'Python', 123 , 4.5, 'Learning', 6.78 ]
mySubList = [901, 'Program']

print (myList)          # print complete list
print (myList[0])       # print first element of the list
print (myList[1:3])     # print elements starting from 2nd till 3rd
print (myList[2:])      # print elements starting from 3rd element
print (mySubList * 2)  # print list twice
print (myList + mySubList) # print concatenated lists
```

List

- Class Activity
  - Display the following results with the list having items ”coffee”, “milk”, ”water”, 35.5, 10.5, 8.0

```
= RESTART:
['coffee', 'milk', 'water', 35.5, 10.5, 8.0]
milk
['water']
[10.5, 8.0]
```

Sequence Type

- Tuples
  - another sequence data type that is similar to the list
  - consists of a number of values separated by commas
  - unlike lists, however, tuples are enclosed within parentheses

- Differences between Lists and Tuples
  - lists are enclosed in brackets ([ ]) and their elements and size can be changed
  - tuples are enclosed in parentheses (( )) and cannot be updated → tuples can be thought of as read-only lists

Tuple

- Example
  - Input the following code and run the file

Programming Code

```
myTuple = ('Python', 123 , 4.5, 'Learning', 6.78)
mySubTuple = (901, 'Program')

print (myTuple)              # print the complete tuple
print (myTuple[0])           # print first element of the tuple
print (myTuple[1:3])         # print elements of the tuple starting from 2nd till 3rd
print (myTuple[2:])          # print elements of the tuple starting from 3rd element
print (mySubTuple * 2)       # print the tuple twice
print (myTuple + mySubTuple) # print concatenated tuples
```

Tuple

- Example
  - Input the following code and run the file

Example.py

File Edit Format Run Options Window Help

```
myTuple = ('Python', 123 , 4.5, 'Learning', 6.78)
mySubTuple = (901, 'Program')

print (myTuple)                # print the complete tuple
print (myTuple[0])             # print first element of the tuple
print (myTuple[1:3])           # print elements of the tuple starting from 2nd till 3rd
print (myTuple[2:])            # print elements of the tuple starting from 3rd element
print (mySubTuple * 2)        # print the tuple twice
print (myTuple + mySubTuple)   # print concatenated tuples
```

Tuple

- Class Activity
  - Display the following results with the tuple having items ”Arts”, “Business”, ”IT”, “Geography”, “Language”

```
= RESTART:
('Arts', 'Business', 'IT', 'Geography', 'Language')
IT
('IT', 'Geography')
('Geography', 'Language')
```

Dictionary

- Dictionary
  - kind of hash table type
  - work like associative arrays or hashes found in Perl and consist of key-value pairs
  - a dictionary key can be almost any Python type, but are usually numbers or strings
  - values, on the other hand, can be any arbitrary Python object
  - dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([])

Dictionary

- Example
  - Input the following code and run the file

Programming Code

```
myDict = {}
myDict['Peter'] = 100
myDict[123]     = "Course"

mySubdict = {'name': 'Mary','Age':18, 'Course': 'IT'}

print (myDict)            # print the complete dictionary
print (myDict['Peter'])       # print value for the key
print (myDict[123])           # print value for the key
print (mySubdict)         # print complete dictionary
print (mySubdict.keys())   # print all the keys
print (mySubdict.values()) # print all the values
```

Dictionary

- Example
  - Input the following code and run the file

Example.py

File Edit Format Run Options Window Help

```
myDict = {}
myDict['Peter'] = 100
myDict[123]     = "Course"

mySubdict = {'name': 'Mary','Age':18, 'Course': 'IT'}


print (myDict)              # print the complete dictionary
print (myDict['Peter'])        # print value for the key
print (myDict[123])            # print value for the key
print (mySubdict)           # print complete dictionary
print (mySubdict.keys())    # print all the keys
print (mySubdict.values())  # print all the values
```

Dictionary

- Class Activity
  - Display the following results with the dictionary having the following items

John	85
Mary	90.5
Peter	100

```
= RESTART:
{'John': 85, 'Mary': 90.5, 'Peter': 100}
90.5
dict_keys(['John', 'Mary', 'Peter'])
dict_values([85, 90.5, 100])
```

Boolean

❑ Boolean

- Boolean type provides two built-in values, True and False
- these values are used to determine the given statement true or false
- denote by the class bool
- true can be represented by any non-zero value or 'T'
- false can be represented by the 0 or 'F'

❑ Example

- Input the following code and run the file

Programming Code

```
print(type(True))
print(type(False))
```

```
>>> print(type(False))
<class 'bool'>
>>> print(type(True))
<class 'bool'>
```

Set

- Set
  - the unordered collection of the data type
  - iterable, mutable(can modify after creation), and has unique elements
  - the order of the elements is undefined → may return the changed sequence of the element
  - set is created by using a built-in function ***set()***, or a sequence of elements is passed in the curly braces and separated by the comma
  - can contain various types of values
- Example
  - Input the following code and run the file

Programming Code

```
mySet = {'Learning','Python'}
print(mySet)
```

Example.py

File Edit Format Run Options Window Help

```
mySet = {'Learning','Python'}
print(mySet)
```

Data Type Conversion

Data Type Conversion

the process of converting a Python data type into another data type

- Implicit Type Conversion
  - when the data type conversion takes place during compilation or during the run time
  - Python handles the implicit data type conversion, so we don't have to explicitly convert the data type into another data type

- Explicit Type Conversion / Typecasting
  - explicit type conversion takes place when the programmer clearly and explicitly defines the same in the program

Data Type Conversion

- Common Explicit Type Conversion Function

Function	Description
int(y [base])	It converts y to an integer, and Base specifies the number base.
float(y)	It converts y to a floating-point number.
str(y)	It converts y to a string.
tuple(y)	It converts y to a tuple.
list(y)	It converts y to a list.
dict(y)	It creates a dictionary and y should be a sequence of (key, value) tuples.

Data Type Conversion

- Example

```
classA = int(input("How many students in Class A? " ))
classB = int(input("How many students in Class B? " ))

total = classA + classB

print("There are " + str(total) + "students in out school.")

print(type(total))
```

Programming Code

```
classA = int(input("How many students in Class A? " ))
classB = int(input("How many students in Class B? " ))
total = classA + classB
print("There are " + str(total) + "students in out school.")
print(type(total))
```

Data Type Conversion

- Class Activity
  - Create a dictionary called product to store the names and prices (including the delivery fees) of product 1 and product 2
  - Ask the user to enter the name and prices of product 1 and product 2 respectively
  - Display the prices including the delivery fees (delivery fee for product 1 is 5 dollars and delivery fee for product 2 is 10.5 dollars)
  - Display the final product list
  - Sample output is shown

```
=========== Product 1===========
Enter the name of product 1: Juice
Enter the price of product 1: 15.5
Price of Juice with delivery fee: 20.5
=========== Product 2===========
Enter the name of product 2: Chocolate
Enter the price of product 2: 20
Price of Chocolate with delivery fee: 30.5
=========== Product List===========
{'Juice': 20.5, 'Chocolate': 30.5}
```

Hope you enjoy the class

Thank you