

19CSE201 :Advanced Programming

Lecture 17-19

Hello World Again!

This time in Python

By
Ritwik M
Assistant Professor(SrGr)
Dept. Of Computer Science & Engg

About the Language

- Python was developed by Guido van Rossum at the National Research Institute for Mathematics and Computer Science in the Netherlands
- High-level language, can do a lot with relatively fewer lines of code
- Fairly popular among high-level languages
- Robust support for object-oriented programming
- Support for integration with other languages
- Runs on an interpreter system, meaning that code can be executed as soon as it is written.
 - This means that prototyping can be very quick.
- Can be treated in a procedural way, an object-oriented way or a functional way.

Why Python?

- **Python is Interpreted**
 - Python is processed at runtime by the interpreter. You do not need to compile your program before executing it.
- **Python is Interactive**
 - You can actually sit at the Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented**
 - Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language**
 - Supports the development of a wide range of applications from simple text processing to WWW browsers to games

Very Basic Stuff

- Running python programs
 - Command line syntax: `python3 filename.py`
- Variables
 - Dynamic data typing
- Printing – `print("HelloWorld!!")`
- Input – returns a string
 - Syntax: `input(prompt)`
Eg. `string = input()`
- Comments
 - Single `#This is a single line comment`
 - Multiline `''' This is a multiline comment'''`

Python Data Types

Category	Type
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range, string
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview

Python Data Types

- variables can hold values, and every value has a data-type.
- Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it.
- The interpreter implicitly binds the value with its type.
- For Example, `a = 5` . The variable `a` holds integer value five and we did not define its type.
- Python interpreter will automatically interpret variables `a` as an integer type.
- Python provides us the `type()` function, which returns the type of the variable passed.

Numeric

- Numeric data type represent the data which has numeric value
 - Integers
 - This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.
 - Float
 - This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
 - Complex Numbers
 - Complex number is represented by complex class. It is specified as (real part) + (imaginary part)j. For example - $2+3j$
- Example

```
a = 5
print("Type of a: ", type(a))
b = 5.0
print("\nType of b: ", type(b))
c = 2 + 4j
print("\nType of c: ", type(c))
```

String

- String literals in python are surrounded by either single quotation marks, or double quotation marks.
 - 'hello' is the same as "hello".
- You can display a string literal with the print() function:
 - ```
print("Hello")
print('Hello')
```
- Assign String to a variable
  - ```
a = "Hello"  
print(a)
```
- Multiline Strings
 - ```
a = """multi
line
strings."""
print(a)
```



# String Cont.

- **Strings are Arrays**

- 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.
  - `a = "Hello, World!"`  
`print(a[1])`

- **Slicing**

- You can return a range of characters by using the slice syntax.
- Specify the start index and the end index, separated by a colon, to return a part of the string.
  - `b = "Hello, World!"`  
`print(b[2:5])`

# String Cont.

- Negative Indexing

- use negative indexes to start the slice from the end of the string:

- `b = "Hello, World!"`  
`print(b[-5:-2])`

- String Length

- To get the length of a string, use the `len()` function.

- `a = "Hello, World!"`  
`print(len(a))`

# String Methods

- The `strip()` method removes any whitespace from the beginning or the end:
  - `a = " Hello, World! "`  
`print(a.strip())` # returns "Hello, World!"
- The `lower()` / `upper()` methods return the string in lower/uppercase:
  - `a = "Hello, World!"`  
`print(a.lower())`
- The `split()` method splits the string into substrings if it finds instances of the separator:
  - `a = "Hello, World!"`  
`print(a.split(","))` # returns ['Hello', ' World!']

# String Methods Cont.

- To check if a certain phrase or character is present in a string, we can use the keywords `in` or `not in`.
  - ```
txt = "The rain in Spain stays mainly in the plain"  
x = "ain" in txt  
print(x)
```
- To concatenate, or combine, two strings you can use the `+` operator.
 - ```
a = "Hello"
b = "World"
c = a + b
print(c)
```

# String Methods Cont.

- Explore these methods as well.
  - `format()`
  - `endswith()`
  - `index()`
  - `join()`
  - `lstrip()`
  - `replace()`
  - `swapcase()`

# List

Don't just read, Try it!!

- Python Lists are similar to arrays in C.
- A list object is an ordered collection of one or more data items and is changeable.
- The list can contain data of different types.
- The items stored in the list are separated with a comma (,) and enclosed within square brackets [ ].
- Example:
  - ```
thislist = ["apple", "banana", "cherry"]  
print(thislist)
```
- Lists have no fixed size and can be expanded or contracted as needed.
- Items in list can be retrieved using the index.
- Lists can be nested just like arrays, i.e., you can have a list of lists

List Cont.

- Access Items

- You access the list items by referring to the index number:

- ```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

- Range of Indexes

- You can specify a range of indexes by specifying where to start and where to end the range.
- When specifying a range, the return value will be a new list with the specified items.

- ```
thislist = ["apple", "banana", "cherry", "orange", "kiwi",  
"melon", "mango"]  
print(thislist[2:5])
```

List Cont.

- Check if Item Exists

- To determine if a specified item is present in a list use the `in` keyword:

- ```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
 print("Yes, 'apple' is in the fruits list")
```

- List Length

- To determine how many items a list has, use the `len()` function:

- ```
thislist = ["apple", "banana", "cherry"]  
print(len(thislist))
```

- The `list()` Constructor

- It is also possible to use the `list()` constructor to make a new list.

- ```
thislist = list(("apple", "banana", "cherry"))
#note:double round-brackets
```



# List Cont.

- Add Items

- To add an item to the end of the list, use the `append()` method:

- ```
thislist = ["apple", "banana", "cherry"]  
thislist.append("orange")  
print(thislist)
```

- To add an item at the specified index, use the `insert()` method:

- ```
thislist = ["apple", "banana", "cherry"]
thislist.insert(1, "orange")
print(thislist)
```

- Remove Item

- There are several methods to remove items from a list – Try them all

- `remove()`
  - `pop()`
  - `del`
  - `clear()`

# Tuple

- A tuple is a collection which is ordered and unchangeable.
- In Python tuples are written with round brackets.

- Example:

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

- Access Tuple Items

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[1])
```

- Negative Indexing

- Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.

- Example:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple[-1])
```

What about a range of indexes? Try it!!

# Tuple Cont.

- **Change Tuple Values**

- Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.
- But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)
print(x)
```

- **Create Tuple With One Item**

- To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

- **The tuple() Constructor**

- It is also possible to use the tuple() constructor to make a tuple.

# Tuple Cont.

- Tuple Methods

- Python has two built-in methods that you can use on tuples.
- `count()`
  - Returns the number of times a specified value occurs in a tuple
- `index()`
  - Searches the tuple for a specified value and returns the position of where it was found
- `del`
  - Tuples are unchangeable, so you cannot remove items from it, but you can delete the tuple completely using `del`

# Set

- A set is a collection which is unordered and unindexed. In Python, sets are written with curly brackets.

- Example:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

- Access Items

- You cannot access items in a set by referring to an index or a key.
- But you can loop through the set items using a `for` loop, or ask if a specified value is present in a set, by using the `in` keyword.
- Example:

```
thisset = {"apple", "banana", "cherry"}
for x in thisset:
 print(x)
```

# Set Cont.

- Add Items

- To add one item to a set use the `add()` method.
- To add more than one item to a set use the `update()` method.
- Example

```
thisset = {"apple", "banana", "cherry"}
thisset.add("orange")
print(thisset)
```

- Add multiple items to a set

- using the `update()` method
- Example:

```
thisset = {"apple", "banana", "cherry"}
thisset.update(["orange", "mango", "grapes"])
print(thisset)
```

# Set Cont.

- Remove Item

- To remove an item in a set, use the `remove()`, or the `discard()` method.

- Using `remove()`

- ```
thisset = {"apple", "banana", "cherry"}  
thisset.remove("banana")  
print(thisset)
```

- Note: If the item to remove does not exist, `remove()` will raise an error.

- Using `discard()`

- ```
thisset = {"apple", "banana", "cherry"}
thisset.discard("banana")
print(thisset)
```

- Note: If the item to remove does not exist, `discard()` will NOT raise an error.

# Set Cont.

- *Additional Operations on Sets*
  - `union()`
  - `update()` - has an additional use
  - `clear()`
  - `set()`
  - `pop()`
  - `intersection()`
  - `symmetric_difference()`



# Dictionary

- A dictionary is a collection which is unordered, changeable and indexed.
- In Python dictionaries are written with curly brackets, and they have keys and values.
- Example:
  - ```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
print(thisdict)
```
- A dictionary can also contain many dictionaries, this is called nested dictionaries.

Dictionary Cont.

- Accessing Items

- You can access the items of a dictionary by referring to its key name, inside square brackets
 - `x = thisdict["model"]`
- There is also a method called `get()` that will give you the same result
 - `x = thisdict.get("model")`

- Change Values

- You can change the value of a specific item by referring to its key name
 - ```
thisdict = {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964
}
thisdict["year"] = 2020
```

# Dictionary Cont.

- Loop Through a Dictionary

- You can loop through a dictionary by using a for loop.
- When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.
- Eg. Print all key names in the dictionary, one by one

- `for x in thisdict:`  
`print(x)`

- Eg. Print all values in the dictionary, one by one

- `for x in thisdict:`  
`print(thisdict[x])`

- You can also use the `values()` method to return values of a dictionary

- `for x in thisdict.values():`  
`print(x)`

- Loop through both keys and values, by using the `items()` method

- `for x, y in thisdict.items():`  
`print(x, y)`

# Dictionary Cont.

- Check if Key Exists

- To determine if a specified key is present in a dictionary use the `in` keyword

```
• thisdict = {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964
}
if "model" in thisdict:
 print("Yes, 'model' is a key in the thisdict dictionary")
```

- To determine how many items (key-value pairs) a dictionary has, use the `len()` function.

```
• print(len(thisdict))
```

# Dictionary Cont.

- Adding Items

- Adding an item to the dictionary is done by using a new index key and assigning a value to it

- ```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict["color"] = "red"  
print(thisdict)
```

- Removing Items

- There are several methods to remove items from a dictionary – try them!

- `pop()`
 - `popitem()`
 - `del`
 - `clear()`

Python Operators

- Python divides the operators in the following groups:
 - Arithmetic operators
 - Assignment operators
 - Comparison operators
 - Logical operators
 - Identity operators
 - Membership operators
 - Bitwise operators

Arithmetic Operators

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

Arithmetic Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
=	x = 3	x = x 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Comparison Operators

Operator	Name	Example
==	Equal	<code>x == y</code>
!=	Not equal	<code>x != y</code>
>	Greater than	<code>x > y</code>
<	Less than	<code>x < y</code>
>=	Greater than or equal to	<code>x >= y</code>
<=	Less than or equal to	<code>x <= y</code>

Logical Operators

Operator	Description	Example
and	Returns True if both statements are true	<code>x < 5 and x < 10</code>
or	Returns True if one of the statements is true	<code>x < 5 or x < 4</code>
not	Reverse the result, returns False if the result is true	<code>not(x < 5 and x < 10)</code>

Identity Operators:

- They are used to check if two values (or variables) are located on the same part of the memory.
- Two variables that are equal does not imply that they are identical.

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Membership Operators:

- Membership operators are used to test whether a value or variable is found in a sequence (string, list, tuple, set and dictionary).
- In a dictionary we can only test for presence of key, not the value.

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Bitwise Operators:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Python Conditional Statements

- If

- ```
a = 33
b = 200
if b > a:
 print("b is greater
than a")
```

- Elif

- ```
a = 33
b = 33
if b > a:
    print("b is greater
than a")
elif a == b:
    print("a and b are
equal")
```

- Else

- ```
a = 200
b = 33
if b > a:
 print("b is greater
than a")
elif a == b:
 print("a and b are
equal")
else:
 print("a is greater
than b")
```

# Python Conditional Statements Cont.

- **Indentation**

- Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.

- **The Pass statement**

- If statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.
- ```
a = 33
b = 200
if b > a:
    pass
```

Exercise 1

- Which of the following is invalid?

-

- a) `_a = 1`

- b) `__a = 1`

- c) `__str__ = 1`

- d) none of the mentioned

-

- Answer: d

Quick Summary

- Variables
- Datatypes
- Declaration
- Operators
- Errors
- Conditional Operators
- Examples
- Exercises

Try out your new skills on HPOJ

URL: <http://hpoj.cb.amrita.edu:8000/>

Up Next

More on Basic Python – functions