Lecture 2 – Python

Mr. Yousif G. Arshak

University of Zakho

Computer Science Department

yousif.arshak@uoz.edu.krd

OUTLINES

Python Introduction

- Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.
- It is used for:
  - ➢web development (server-side),
  - ➢software development,
  - ➢mathematics,
  - ➢system scripting.

Comments

```
#This is a comment
print("Hello, World!")

print("Hello, World!") #This is a comment

"""
This is a comment
with more than just one line
"""
print("Hello, World!")
```

Variables

- Variables are containers for storing data values. Unlike other programming languages, Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.
- Example

```
a = 10
b = “My Name is .. "
print(x)
print(y)
```

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

```
x = 2 # x is of type integer
x = 2.2 # x is of type float
x = “I am a string" # x is now of type string
x = ’I am also a string’ # x is now of type string
print(x)
```

Legal and Illegal variable names

```
#Legal variable names:
myvar = “I’m a string variable"
my_var = "I’m a string variable"
_my_var = "I’m a string variable"
myVar = "I’m a string variable"
MYVAR = "I’m a string variable"
myvar2 = "I’m a string variable"

#Illegal variable names:
2myvar = “number can’t be at the beginning"
my-var = “minus sign can’t be used to define variable”
my var = ”space can’t be used to define variable”
```

Assign Value to Multiple Variables

- Python allows you to assign values to multiple variables in one line:

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

- You can assign the *same* value to multiple variables in one line:

```
x = y = z = "Orange"
print(x)
print(y)
print(z)
```

Data Types

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

Getting the Data Type

- You can get the data type of any object by using the type() function:
- Example:

```
x = 5
print(type(x))
Output: <class 'int'>
```

Setting the Data Type

- In Python, the data type is set when you assign a value to a variable:

Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana", "cherry"})`	frozenset
`x = True`	bool
`x = b"Hello"`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	Memory view

If you want to specify the data type, you can use the following constructor functions:

Example	Data Type
`x = str("Hello World")`	str
`x = int(20)`	int
`x = float(20.5)`	float
`x = complex(1j)`	complex
`x = list(("apple", "banana", "cherry"))`	list
`x = tuple(("apple", "banana", "cherry"))`	tuple
`x = range(6)`	range
`x = dict(name="John", age=36)`	dict
`x = set(("apple", "banana", "cherry"))`	set
`x = frozenset(("apple", "banana", "cherry"))`	frozenset
`x = bool(5)`	bool
`x = bytes(5)`	bytes
`x = bytearray(5)`	bytearray
`x = memoryview(bytes(5))`	Memory view

Python Numbers

- There are three numeric types in Python:
  - int
  - float
  - Complex
-
```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

Convert from one type to another:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)
```

Note: You cannot convert complex numbers into another number type.

Random Number

- Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:
- Import the random module, and display a random number between 1 and 9:

```
import random

print(random.randrange(1, 10))
```

Python Strings

- String literals in python are surrounded by either single quotation marks, or double quotation marks.
- `'hello'` is the same as `"hello"`.

- Get the character at position 1 (remember that the first character has the position 0):

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

Slicing

- Specify the start index and the end index, separated by a colon, to return a part of the string.
- Example: Get the characters from position 2 to position 5 (not included):

  -
    ```
    b = "Hello, World!"
    print(b[2:5])
    ```
- Negative Indexing: Use negative indexes to start the slice from the end of the string:
- Get the characters from position 5 to position 1 (not included), starting the count 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.
- Example: The len() function returns the length of a string:

  -
    ```
    a = "Hello, World!"
    print(len(a))
    ```
    ```
    Output: 13
    ```

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() method returns the string in lower case:

  -
    ```
    a = "Hello, World!"
    print(a.lower())
    ```
- The upper() method returns the string in upper case:

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

String Methods

- The replace() method replaces a string with another string:

  -
    ```
    a = "Hello, World!"
    print(a.replace("H", "J"))
    ```
- The split() method splits the string into substrings if it finds instances of the separator:

  -
    ```
    a = "Hello, World!"
    print(a.split(",")) # returns ['Hello', ' World!']
    ```

Check String

- To check if a certain phrase or character is present in a string, we can use the keywords in or not in.
- Example: Check if the phrase "ain" is present in the following text:

  -
    ```
    txt = "The rain in Spain stays mainly in the plain"
    x = "ain" in txt
    print(x) # True
    ```
- Check if the phrase "ain" is NOT present in the following text:

  -
    ```
    txt = "The rain in Spain stays mainly in the plain"
    x = "ain" not in txt
    print(x) # False
    ```

String Concatenation

- To concatenate, or combine, two strings you can use the + operator.

Example: Merge variable a with variable b into variable c:

```
a = "Hello"
b = "World"
c = a + b
print(c) # HelloWorld
```

- To add a space between them, add a " ":

```
a = "Hello"
b = "World"
c = a + " " + b
print(c) # Hello World
```

String Format

- As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

```
age = 36
txt = "My name is John, I am " + age
print(txt) #TypeError: must be str, not int
```

But we can combine strings and numbers by using the `format()` method!

The `format()` method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

- Example
- Use the format() method to insert numbers into strings:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age)) #My name is John, and I am 36
```

- The format() method takes unlimited arguments

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
 #I want 3 pieces of item 567 for 49.95 dollars.
```

Escape Character

- To insert characters that are illegal in a string, use an escape character.
- Example: You will get an error if you use double quotes inside a string that is surrounded by double quotes:
  - `txt = "We are the so-called "Vikings" from the north.“`
  - `#SyntaxError: invalid syntax`
- To fix this problem, use the escape character \":
  - `txt = "We are the so-called \"Vikings\" from the north.“`
  - `# We are the so-called "Vikings" from the north.`

Python Booleans

- Booleans represent one of two values: True or False.
- When you compare two values, the expression is evaluated and Python returns the Boolean answer:

```
print(10 > 9)  # Ture
print(10 == 9) # False
print(10 < 9)  # False
```

- When you run a condition in an if statement, Python returns `True` or `False`:

Example Print a message based on whether the condition is `True` or `False`:

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

```
#b is not greater than a
```

List

- A list is a collection which is ordered and changeable. In Python lists are written with square brackets.
- Example: Create a List:

  -
    ```
    mylist = ["apple", "banana", "cherry"]
    print(mylist) #['apple', 'banana', 'cherry']
    ```
- Access Items of the list: You access the list items by referring to the index number:
- Example: Print the second item of the list:

  -
    ```
    mylist = ["apple", "banana", "cherry"]
    print(mylist[1]) # banana
    ```
- Negative Indexing means beginning from the end, `-1` refers to the last item, `-2` refers to the second last item etc.

- Example: Print the last item of the list:

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

List

- Range of Indexes: You can specify a range of indexes by specifying where to start and where to end the range
- Example: Return the third, fourth, and fifth item:

  -
    ```
    mylist =
    ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
    print(mylist[2:5]) # ['cherry', 'orange', 'kiwi’]
    ```
- Change Item Value: To change the value of a specific item, refer to the index number:
- Example: Change the second item:

  -
    ```
    mylist = ["apple", "banana", "cherry"]
    mylist[1] = "blackcurrant"
    print(mylist) # ['apple', 'blackcurrant', 'cherry']
    ```

Tuple

- A tuple is a collection which is ordered and **unchangeable**. In Python tuples are written with round brackets ().
- Example: Create a Tuple:

  -
    ```
    mytuple = ("apple", "banana", "cherry")
    print(mytuple) # ('apple', 'banana', 'cherry
    ```
- Access Tuple Items: You can access tuple items by referring to the index number, inside square brackets:

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

Python Dictionaries

- 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: Create and print a dictionary:

  -
    ```
    mydict = {
      "brand": "Ford",
      "model": "Mustang",
      "year": 1964
    }
    print(mydict) #{'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
    ```
- Example: Get the value of the "model" key:
  - `x = mydict["model"] # Mustang`

  There is also a method called `get()` that will give you the same result:
  x = mydict.get("model") `# Mustang`
- To change values:
  ```
  mydict["year"] = 2018
  ```

Python Operators

- Operators are used to perform operations on variables and values.
- Python divides the operators in the following groups:
  - Arithmetic operators (+,-,*,/,%,**,//)
  - Assignment operators (=, +=, -=, *=, /=, **=, //=, %=, &=, ^=, >>=, <<=)
  - Comparison operators (==, !=, <, >, <=, >=)
  - Logical operators (and, or, not)
  - Identity operators (is, not)
  - Membership operators (in, not in)
  - Bitwise operators (&, |, ^, ~, <<, >>)

For more info check out this link: https://www.w3schools.com/python/python_operators.asp

Arithmetic operators

```
x = 2
y = 5
print(x + y) # 7
print(x ** y) #same as 2*2*2*2*2
```

Assignment operators

```
x = 5
print(x) # 5
x += 3
print(x) # 8
x = 5
x >>= 2   #shift right with 2 bits
print(x) # 1
```

Python Comparison Operators

```
x = 5
y = 3
print(x == y) # returns False because 5 is not equal to 3
```

Python Logical Operators

```
x = 5
print(x > 3 and x < 10) # returns True because 5 is greater than 3 AND 5 is less than 10
print(x > 3 or x < 4) # returns True because one of the conditions are true (5 is greater than 3, but 5 is not less than 4)
print(not(x > 3 and x < 10)) # returns False because not is used to reverse the result
```

Python Identity Operators

```
x = ["apple", "banana"]
y = ["apple", "banana"]
z = x
print(x is z) # returns True because z is the same object as x
print(x is y) # returns False because x is not the same object as y, even if they have the same
content
print(x == y) # to demonstrate the difference betweeen "is" and "==": this comparison returns
True because x is equal to y

print(x is not z) # returns False because z is the same object as x

print(x is not y) # returns True because x is not the same object as y, even if they have
the same content

print(x != y) # to demonstrate the difference betweeen "is not" and "!=": this
comparison returns False because x is equal to y
```

Python Membership Operators

```
x = ["apple", "banana"]

print("banana" in x) # returns True because a sequence with the value "banana" is in the list

print("pineapple" not in x) # returns True because a sequence with the value "pineapple" is not in the list
```

Python Bitwise Operators

Bitwise operators are used to compare (binary) numbers:

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