COMP07027 Introduction to Programming

School of Computing, Engineering and Physical Sciences Scotland Academy Wuxi

Dr Muhammad Aslam
Lecturer UWS Wuxi
Muhammad.Aslam@uws.ac.uk

Lecture 2: Data and Variables

Agenda of the lecture 1

01 Python Variable

02 Python Data Type

03 Python Operators

1-Python Variables

Creating Variables

- Variables work like a container, which provides ability to a python program to store a data value.
- There is no specific command in Python to declare a variable, variable is created by default once we assign a value to a variable.

```
x = 6
y = “Xiaopeng”
z = “Dengpan”
print(x)
print(y)
print(z)
```

- We do not need to declare variable with any particular *type*, but we have the ability to type the variable type and can also change the variable type.

```
x = 77          # x is of type int
x = “Hello" # x is now of type str
print(x)
```

What is the need of variables

In computer programming it is the basic requirement to able to generate variables to store the data, and perform computational tasks over these variables.

We achieve this with help of variables.

Have a look at this program …. If we need to add three variables with values of 12, 13, and 14.

```
x = 12
y = 13
z = 14

A=x+y+z

print(A)
```

Casting of Variables

We can specify the data type, although we do not need particularly.

```
x = str(6)    # x will be ‘6'
y = int(5)    # y will be 5
z = float(8)  # z will be 8.0
```

- The type() function can be used to get the type of the variables given in program. Example

```
x = 23
y = “Hayat”
print(type(x))
print(type(y))
```

- String variables can be declared either by using single or double quotes:

```
x = “Jin"
# is the same as
x = ’Jin'
```

Rules for Python Variable Names

- The name of the variable can have a short name or a more descriptive name.
- Basic Rules of creating Variables:

1. A variable name must start with a letter or the underscore character
2. A variable name cannot start with a number
3. A variable name can only contain alpha-numeric characters and underscores.
4. Variable names are case-sensitive

```
myname = “Aslam"
my_name = “Aslam"
_my_name =“Aslam"
myName = “Aslam"
MYNAME = “Aslam"
myvar2 =  “Aslam“
```

Multiple Variable Names

•**Multiple name variable:** variable names with more than one word can be difficult to read, there are several techniques you can use to make them more readable:

•Camel Case:
myVariableName="Xiaopeng"

- Pascal Case:

MyVariableName="Xiaopeng"

- Snake Case:

My_variable_name="Xiaopeng"

Many Values to Multiple Variables

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

```
x, y, z, a = “red", “yellow", “pink“, “white”
print(x)
print(y)
print(z)
print(a)
```

- One Value to Multiple Variables: you can assign the *same* value to multiple variables in one line:

```
x = y = z =  “white”
print(x)
print(y)
print(z)
```

Many Values to Multiple Variables

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

```
x, y, z, a = “red", “yellow", “pink“, “white”
print(x)
print(y)
print(z)
print(a)
```

- One Value to Multiple Variables:

And you can assign the *same* value to multiple variables in one line:

```
x = y = z =  “white”
print(x)
print(y)
print(z)
```

- Unpack a Collection

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

Python - Output Variables

Output Variables: Python use print statement is often used to output variables.
To combine both text and a variable, Python uses the + character

```
x = "awesome"
print("Python is " + x)
```

You can also use the + character to add a variable to another variable:

```
x = "Python is "
y = "awesome"
z =  x + y
print(z)
```

For numbers, the + character works as a mathematical operator:

```
x = 5
y = 10
print(x + y)
```

Used defined Input Variable

- Input function is used to defined a variable, in which the value is given by the user in the console:

```
x=input()
y=input()
Z=x+y
Print(x+y)
```

- This program will ask user to put information of x, and y. To make it obvious we can notify user with a message:

```
x=input("Add the value of first number")
y=input("Add the value of second number")
Z=x+y
Print(x+y)
```

- By default value of input is a string value, we have to do casting of input variable

```
x=int(input("Add the value of first number"))
y=int(input("Add the value of second number"))
Z=x+y
Print(x+y)
```

1-Python Data Types

Python Built-in Data Types

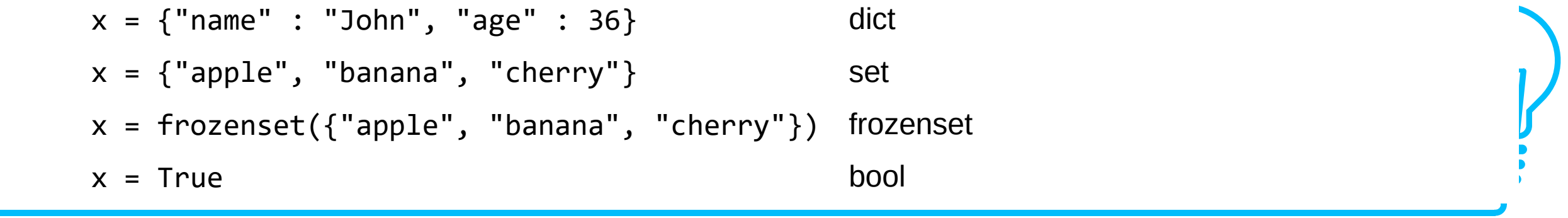
- Variables can store data of different types, and different types can do different things.

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

Setting Data Type

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

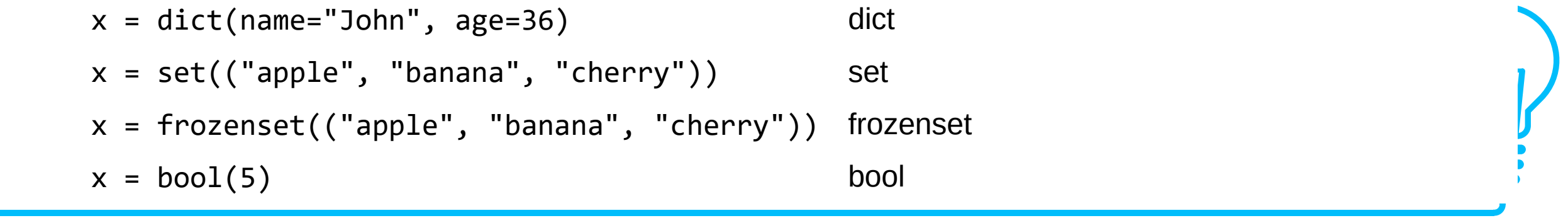
Example	Data Type
`x = "Hello World"`	str
`x = 20`	int
`x = 20.5`	float
`x = 1j`	complex
`x = ["apple", "banana", "cherry"]`	list
`x = ("apple", "banana", "cherry")`	tuple
`x = range(6)`	range
`x = {"name" : "John", "age" : 36}`	dict
`x = {"apple", "banana", "cherry"}`	set
`x = frozenset({"apple", "banana",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



Setting the Specific Data Type

- 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



Data types

There are three numeric types in Python:

- int
- float
- Complex

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

- To verify the type of any object in Python, use the type() function:

```
print(type(x))
print(type(y))
print(type(z))
```

Int Data type

- Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

```
x = 1
y = 35656222554887711
z = -3255522
print(type(x))
print(type(y))
print(type(z))
```

float Data type

- Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

- ```
  x = 1.10
  y = 1.0
  z = -35.59

  print(type(x))
  print(type(y))
  print(type(z))
  ```

- Float can also be scientific numbers with an "e" to indicate the power of 10.
- ```
  x = 35e3
  y = 12E4
  z = -87.7e100

  print(type(x))
  print(type(y))
  print(type(z))
  ```

Complex Data type

- Complex: Complex numbers are written with a "j" as the imaginary part:
- ```
  x = 3+5j
  y = 5j
  z = -5j
  print(type(x))
  print(type(y))
  print(type(z))
  ```


  ```

Type conversion

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

Random number

- 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 random

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


  ```

Strings Data Type

- Strings: Strings 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’)
```

- Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

```
a = "Hello"
print(a)
```

- You can assign a multiline string to a variable by using three quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

Strings Data Type

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

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

- Looping Through a String: Since strings are arrays, we can loop through the characters in a string, with a for loop.

```
for x in "banana":
  print(x)
```

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

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

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

```
txt = "The best things in life are free!"
print("free" in txt)
```

- Use it in an if statement:

```
txt = "The best things in life are free!"
if "free" in txt:
  print("Yes, 'free' is present.")
```

Python - Slicing Strings

- 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.
- Get the characters from position 2 to position 5 (not included):

- ```
  b = "Hello, World!"
  print(b[2:5])
  ```
- Slice From the Start: By leaving out the start index, the range will start at the first character:
- ```
  b = "Hello, World!"
  print(b[:5])
  ```
- Slice To the End: By leaving out the *end* index, the range will go to the end:
- Get the characters from position 2, and all the way to the end:
- ```
  b = "Hello, World!"
  print(b[2:])
  ```


  ```

Python - Modify Strings

- Python - Modify Strings: Python has a set of built-in methods that you can use on strings.
- The upper() method returns the string in upper case:

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

- The lower() method returns the string in lower case:

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

- The strip() method removes any whitespace from the beginning or the end:

```
a = " Hello, World! “
print(a.strip()) # returns "Hello, World!"
```

- 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!']
```

Python - String Concatenation

- String Concatenation: To concatenate, or combine, two strings you can use the + operator.
- ```
  a = "Hello"
  b = "World"
  c = a + b
  print(c)
  ```
- To add a space between them, add a " ":
- ```
  a = "Hello"
  b = "World"
  c = a + " " + b
  print(c)
  ```

- * creates multiple occurrences of a string

Python Booleans

- Boolean Values In programming you often need to know if an expression is True or False. You can evaluate any expression in Python, and get one of two answers, True or False. When you compare two values, the expression is evaluated and Python returns the Boolean answer:

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

- 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")
```

1-Python Operators

Python Operators

Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators

Python 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

Order of precedence

Expressions are performed in the following order

1. expressions in brackets
2. expressions involving **
3. expressions involving * / %
4. expressions involving + -

Order of precedence

So,	2 * 7 – 4	=	10	
	2 * (7 – 4)	=	6	
	2 ** 3 + 1		=	9
	2 ** (3 + 1)	=	16	

Evaluate these in Python to check

Python Assignment 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

Python Comparison Operators

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

Python Logical Operators

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

Python Identity Operators

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

Python Membership Operators

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

Thank You