CSC101 – Introduction to ICT
Lecture 04
Elementary Programming

Course Instructors

Dr. Majid Iqbal **Dr. Ashfaq Hussain Farooqi** **Ms. Sajida Kalsoom** **Mr. Rizwan Rashid**

Lecture Outline

- More about Print() Function
- Operators and operands
- Evaluating Expressions and Operator Precedence
- Augmented Assignment Operators
- Data Types in Python
- Type Conversion Functions

The `print` Function

- `print` function: displays output on the screen
- Argument: data given to a `print` function
  - Example: data that is printed to screen
- Python allows one to display multiple items with a single call to print
  - Items are separated by commas when passed as arguments
  - Arguments displayed in the order they are passed to the function
  - Items are automatically separated by a space when displayed on screen

Example1

```
Print("Hello World")
Print(8)
Print(9.9)
```

Example1 Output

```
Hello World
8
9.9
```

Example-2

```
Print("Hello World",8,9.9)
```

Example2-Output

```
Hello World 8 9.9
```

The `print` Function

```
# Create variables to reference two strings.
first_name = 'Kathryn'
last_name = 'Marino'

# Display the values referenced by the variables.
print(first_name, last_name)
```

Program Output

```
Kathryn Marino
```

More About Data Output

- Suppressing the `print` Function's Ending Newline
  - Newline character at end of printed data
  - Special argument **`end='delimiter'`** causes `print` to place *`delimiter`* at end of data instead of newline character

Example- Normal print()

```
print('One')
print('Two')
print('Three')
```

Output

```
One
Two
Three
```

Example- print() with end=' '

```
print('One', end=' ')
print('Two', end=' ')
print('Three')
```

Output

```
One  Two  Three
```

More About Data Output

- Specifying an Item Separator
- `print` function uses space as item separator
  - Special argument **`sep='delimiter'`** causes `print` to use *`delimiter`* as item separator

Example- print() -Normal

```
print('One', 'Two‘, 'Three')
```

Output

```
One  Two Three
```

Example- print() with sep=‘ ‘

```
print('One', 'Two‘, 'Three‘,sep = ‘->’)
```

Output

```
One ->Two->Three
```

Operators and operands

- **Operators** are special tokens that represent computations like addition, multiplication and division
- The **values** the operator uses are called operands.

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

Operators and operands

The **/** operator performs a float division that results in a floating number

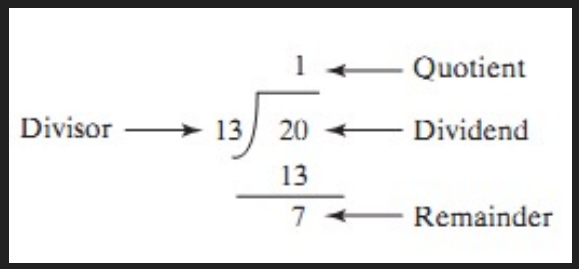
```
>>> 4 / 2
2.0
>>> 2 / 4
0.5
>>>
```

The **//** operator performs an integer division; the result is an integer, and any fractional part is truncated.

```
>>> 5 // 2
2
>>> 2 // 4
0
>>>
```

- The **%** operator, known as *remainder* or *modulo* operator, yields the remainder after division.
  - Example: 20%13

Dividend Divisor



Evaluating Expressions and Operator Precedence

- *Python expressions are evaluated in the same way as arithmetic expressions.*

$$\frac{3 + 4x}{5} - \frac{10(y - 5)(a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

can be translated into a Python expression as:

```
(3 + 4 * x) / 5 - 10 * (y - 5) * (a + b + c) / x +
9 * (4 / x + (9 + x) / y)
```

Evaluating Expressions and Operator Precedence

1. Exponentiation (**) is applied first.
2. Multiplication (*), float division (/), integer division (//) , and remainder operators (%) are applied next.
   - If an expression contains several multiplication, division, and remainder operators, they are applied from left to right.
3. Addition (+) and subtraction (-) operators are applied last.
   - If an expression contains several addition and subtraction operators, they are applied from left to right.

Evaluating Expressions and Operator Precedence

```
3 + 4 * 4 + 5 * (4 + 3) - 1
                        (1) inside parentheses first
3 + 4 * 4 + 5 * 7 - 1
            (2) multiplication
3 + 16 + 5 * 7 - 1
             (3) multiplication
3 + 16 + 35 - 1
  (4) addition
19 + 35 - 1
   (5) addition
    54 - 1
        (6) subtraction
    53
```

Parenthesis
Power
Multiplication
Addition
Left to Right

Augmented Assignment Operators

addition assignment operator

count = count + 1 count **+=** 1

Operator	*Name*	*Example*	*Equivalent*
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Float division assignment	`i /= 8`	`i = i / 8`
`//=`	Integer division assignment	`i //= 8`	`i = i // 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`
`**=`	Exponent assignment	`i **= 8`	`i = i ** 8`

Numeric Data Types (`int` & `float`)

- Data types: categorize value in memory
  - e.g., int for integer, float for real number,
- Numeric literal: number written in a program
  - No decimal point considered int, otherwise, considered float
  - Example: 20, 3.4, -9

Str Data Type

- String is collection of characters.
  - Must be enclosed in single (‘) or double (“) quote marks
- Python can also handle this type of data
  - When string type of data is stored in memory it is classified as str
- The type of data specifies what kind of operations may be applied to it

TypeExample

```
1 first = 5
2 second = 7
3 print('5 x 7:',first * second)
4 first = '5‘
5 second = '7‘
6 print(first * second)
```

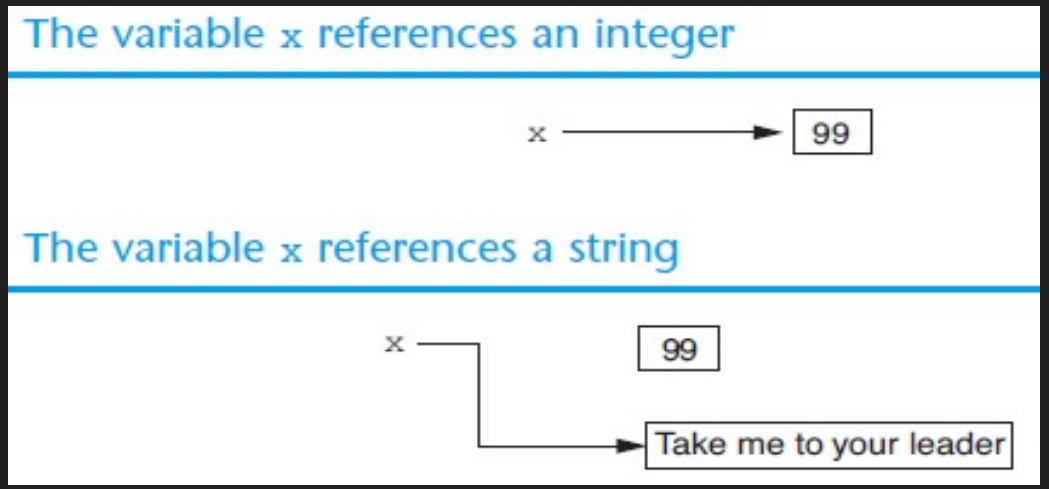
TypeExample-Output

```
5 x 7: 35
Traceback (most recent call last):
  File "main.py", line 6, in <module>
    print(first * second)
TypeError: can't multiply sequence by non-int of type 'str'
```

Reassigning a Variable to a Different Type

- Variables can reference different values while program is running
- A variable can refer to item of any type
  - Variable that has been assigned to one type can be reassigned to another type

```
>>> x = 99
>>> print(x)
99
>>> x = 'Take me to your leader'
>>> print(x)
Take me to your leader.
>>>
```



Type Conversion

- Data type resulting from math operation depends on data types of operands
  - Two `int` values: result is an `int`
  - Two `float` values: result is a `float`
  - `int` and `float`: `int` temporarily converted to `float`, result of the operation is a `float`
    - So, **3 * 4.5** is the same as **3.0 * 4.5**. The result will be float which is 13.5

Type Conversions and Rounding

- **`Int()` Function:** It convert a number/float or integer string like int('10') to an integer or return 0 if no argument is given
  - int("34") returns 34, int(5.6) returns 5
- **`float()`function:** It convert a number or integer string like float('2') to the floating point number
  - float(45) returns 45.0, float('2') returns 2.0
- **`str()` function:** create a new string from given value passed as argument
  - Str(9)returns '9', str(9.9) returns '9.9'
- **`Type()`** function: it returns the type, the value belongs to
  - Type(34) returns <class 'int'>, type(34.5) returns <class 'float'>, type('34.4') returns <class 'str'>
- **`round()` function:** It round a number to the nearest whole value.
  - Round(5.6) return 6, round(4.4) return 4

Summary

- `Print()` Function
- Math Operators and their Precedence
- Evaluation of Expression
- Augmented Assignment Operators
- Data Types and Type Conversion Functions

Next Lecture

- More Built-In Functions
- Math Functions
- More about String
- Formatting output