CS65: Introduction to Computer Science

Expressions
Data Types
Input Function
Division Operator

Md Alimoor Reza
Assistant Professor of Computer Science

Announcements

Code examples available via blackboard

Lab #1 due date

Due date (section#1 + section#2): by tonight on Tuesday, September 16th

Due date (section#3): by tomorrow night on Wednesday, September 17th

Read Python Values and Types *[Very short]*

https://greenteapress.com/thinkpython2/html/thinkpython2002.html#sec10

Quick Review

Recap

- Displaying a text
- Comments in Python
- Variables
- Assignment statements

Recap: Print Statement

The `print` statement is a Python line of code that will output a value when the program is run

The value output will be whatever is between the parentheses ()

hello_world.py

```
1  # name: Alimoor Reza
2  # date: September 16th, 2025
3  # description: my first program in Python programming language
4
5  print("Hello Qingdao University!")
```

In Python, any sequence of characters between double-quotes (“) or single-quotes (‘) is known as a string

String examples:

- **“Hello World”**
- **‘123 ABC’**
- **“Greetings, What is your name?”**

Recap: Comments

Comments are notes of explanation that document lines or sections of a program

Comments are part of the program, but the Python *interpreter* ignores them

They are intended for people who may be reading the source code

- Like your professor
- Or teammates

Python comments are denoted with a #

Recap: Comments

comments.py

comments.py

```
1  # Alimoor Reza
2  # you should always put your name in the comments at the top of your code
3
4  print('this is a string') # comments can go after the lines of code, too
5  print("This, too, is a string")
```

```
# This is a comment

'''This is also a comment'''
""" Yes,
    this
    is
    also
    a
    comment """
```

Recap: Variables

A **variable** is a *name* that represents a value stored in the computer's memory

Programmers use variables to hold values that might change

They're kind of like cells in a spreadsheet

Recap: variables and assignment operator

- Need to use assignment operator (**=**) to store a value
- Location of assignment on the left
- Single value or some calculated value on the right
- **variable_name** = value

```
33  time_sec = 60
34  temp_degree = 27
35
36  mile_to_kilometer = 1.609
37  price_in_dollars = 1500.89
```

Numbers

```
first_name = "Md Alimoor"
last_name = "Reza"
```

Textual data

Recap: Printing text and variables together

We've seen printing out text by itself and variables by themselves. What if I want both in the same print statement?

You can pass multiple things to **`print()`** using *commas* between them.

variables3.py

```
1  # author: Alimoor Reza
2  # description: introduction to variables
3
4  number_of_eggs = 12
5  print("There are ", number_of_eggs, " in the basket.")
6
```

Shell

```
There are  12  in the basket.
>>> %Run variables3.py
There are  12  in the basket.
>>>
```

Recap: Rules for Variable Naming

- Give meaningful variable name to make it easily readable/memorable

```
x = 1.609
```

Vs

```
mile_to_kilometer = 1.609
```

- Name should begin with a lowercase letter

- Use underscore to connect multiple words
  - do not use SPACE in between words

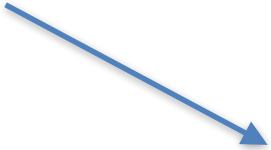
```
milestokilometer = 1.609
MilesToKilometer = 1.609
milesToKilometer = 1.609
```

Vs

```
mile_to_kilometer = 1.609
```

Rules for Variable Naming

- Names can only contain letter, numbers, and underscores
- First character must be a letter or an underscore
  - After that, you can use letter/numbers/underscore
- Cannot be a Python keyword



```
and       del       from      not       while
as        elif      global    or        with
assert    else      if        pass      yield
break     except    import    print
class     exec      in        raise
continue  finally   is        return
def       for       lambda    try
```

- Cannot contain spaces
- Variable names are case sensitive
  - Uppercase and lowercase name will signify different variable

Recap: Assignment Statements

Code with a = is called an *assignment statement*

You use an *assignment statement* to create a variable and make it reference a piece of data

Examples:

```
age = 25
hourly_rate = 15.5
greeting = "Hello everyone"
```

Whatever is on the right side of = will be evaluated and saved to the variable named on the left side.

Questions?

New Material

Roadmap

- Expressions
- Data types
- Receiving input from user
  - Counterpart of showing an user some outputs
- Division operators

Expressions

An *expression* is something evaluated by the Python interpreter

```
num_students = 42
num_adults = 10
total_field_trip_lunches = num_students + num_adults
print("Total field trip lunches:", total_field_trip_lunches)
```

Expression

A fragment of Python code that calculates a new value called an expression

For example, you can convert miles into meters using the following expression:

```
num_of_miles = 10
miles_to_kilometer = 1.609
```

```
num_of_meter = num_of_miles*miles_to_kilometer*1000
```

Expressions

An *expression* is something evaluated by the Python interpreter

expression.py

```
expression.py
1  # author: Alimoor Reza
2  # description: introduction to expression
3
4  num_of_students = 42
5  num_of_adults = 10
6  total_field_trip_lunches = num_of_students + num_of_adults
7  print("Total field trip lunches ", total_field_trip_lunches)
```

When `num_of_students + num_of_adults` is evaluated, it results in an answer, so it is an *expression*

Expressions

An *expression* is something evaluated by the Python interpreter

expression.py

```
# author: Alimoor Reza
# description: introduction to expression

num_of_students = 42
num_of_adults = 10
total_field_trip_lunches = num_of_students + num_of_adults
print("Total field trip lunches ", total_field_trip_lunches)
```

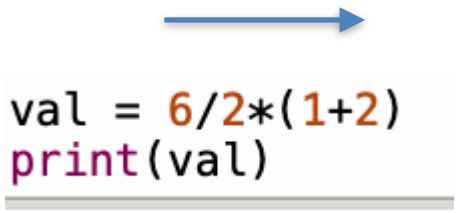
When **`num_of_students + num_of_adults`** is evaluated, it results in an answer, so it is an *expression*

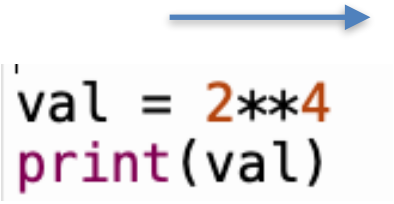
Expressions include operators, like +, and operands like `num_of_students` and `num_of_adults`

Arithmetic Operators

Python supports all of the arithmetic operators you know and love:

- ( ) parentheses
- ** exponents
- * multiplication
- / division
- + addition
- - subtraction



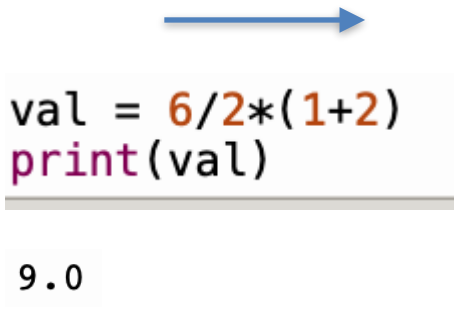


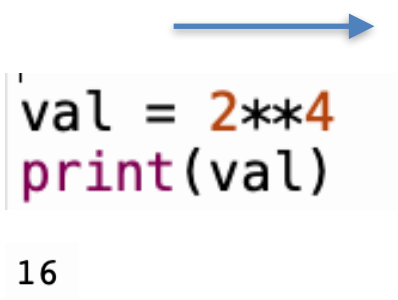
PEMDAS order of operations

Arithmetic Operators

Python supports all of the arithmetic operators you know and love:

- ( ) parentheses
- ** exponents
- * multiplication
- / division
- + addition
- - subtraction





PEMDAS order of operations

Arithmetic Operators

Python supports all of the arithmetic operators you know and love:

- ( ) parentheses
- ** exponents
- * multiplication
- / division
- + addition
- - subtraction

```
val = 6/2*(1+2)+5
print(val)
```

PEMDAS order of operations

Let’s try it

Exercise

1. Open Thonny
2. Create a new File
3. Save it as “day03_exercise.py”

You may want to consider putting this into its own folder inside your CS65 folder

Here are some example exercises!

- Can you compute the area of a rectangle?
  - Length of the two sides will be given in variables

- Can you compute the area of a circle?
  - Radius of the circle is given
  - Value of Pi is 3.14159

Roadmap

- Expressions
- Data types
- Receiving input from user
  - Counterpart of showing an user some outputs
- Division operators

Types

All data has a ***type*** which defines what kind of thing it is and what you can do with it.

Types

All data has a ***type*** which defines what kind of thing it is and what you can do with it.

Types we've seen so far:

```
3
4 num_students = 42 # whole numbers
5 avg_gas_estimate = 2.70 # numbers with decimal
6 print("Hello world!") # text data (strings)
```

Names of some types

integer: whole numbers

Examples: 4, 144, -235

floating point: numbers with decimal points

Examples: 2.56, -2.4, 5.0

string: text data

Examples: “hello world”, “nice to meet you”

The `type()` function can be used to tell you what kind of type something is.

Both values and variables can have types.

What will be output?

types.py

types.py

```
1  # Alimoor Reza
2  # Introduction to data types
3
4  num_students = 42
5  avg_gas_estimate = 3.15
6  print("Hello Qingdao students")
7
8  print(type(5.5))
9  print(type(avg_gas_estimate))
10 print(type(num_students))
```

Intro to REPL

A Read-Eval-Print Loop, or **REPL**, is a computer environment where user inputs are read and evaluated, and then the results are returned to the user.

Intro to REPL

A Read-Eval-Print Loop, or **REPL**, is a computer environment where user inputs are read and evaluated, and then the results are returned to the user.

```
8 print(type(2.70))
```

Shell

```
>>> %Run Types.py
 <class 'float'>
```

```
>>> type(2.70)
<class 'float'>
```

Expressions on strings

We've seen how arithmetic operators work on numerical types like ints and floats.

Strings have their own operators too - and some look like arithmetic operators!

Expressions on strings

We've seen how arithmetic operators work on numerical types like ints and floats.

Strings have their own operators too - and some look like arithmetic operators!

string_expression.py

```
1  # Alimoor Reza
2  # string expressions
3
4  greeting = "Hello "
5  name = "Professor Reza"
6  print(greeting + name)
```

What do you think will happen?

string_expressions.py

Expressions on strings

We've seen how arithmetic operators work on numerical types like ints and floats.

Strings have their own operators too - and some look like arithmetic operators!

string_expression.py

```
1  # Alimoor Reza
2  # string expressions
3
4  greeting = "Hello "
5  name = "Professor Reza"
6  print(greeting + name)
```

```
>>> %Run string_expression

  Hello Professor Reza

>>>
```

What about this?

string_expression.py

```
1  # Alimoor Reza
2  # string expressions
3
4  greeting = "Hello "
5  name = "Professor Reza"
6  print(greeting + name)
7
8  print(greeting - name) # what will happen here?
```

string_expressions.py

Other Operations

The `len` function will output the number of characters that a string has

```
>>> len("hello")
5
>>> len("123")
3
>>> len("oh my gosh this is a very long string")
37
```

Numerical Formats

Python has two different forms of numbers:

- `int` – short for integer. These are whole numbers that are not a fraction. Examples are 0, 4, 24, -12
- `float` – floats are numbers with a decimal point. float is short for floating-point numbers. Examples include 1.3, -34.5, 5.0

Other Operations

The `int` function will convert the number/string into an integer number

other_functions.py

```
1  # alimoor reza
2  # int() function and float() function
3
4  int_num = int(4.5)
5  print(int_num)
```

4

The `float` function will convert a number/string into an decimal number

other_functions.py

```
1  # alimoor reza
2  # int() function and float() function
3
4  float_num = float(4)
5  print(float_num)
```

4.0

Other Operations

The `int` function will convert the number/string into an integer number

The `float` function will convert a number/string into an decimal number

```
>>> int(4.5)
4
>>> float(4)
4.0
>>> float("3.4")
3.4
```

What do you think will happen here?

types2.py

types2.py

```
1  # Alimoor Reza
2  # Introduction to data types
3
4  val = "32"
5  print( type(val) )
6  print( type( len(val) ) )
7
```

What do you think will happen here?

types3.py

```
1  # Alimoor Reza
2  # Introduction to data types
3
4  val = "32"
5
6  # what do you think will happen for the following?
7  print(val)
8  print( int(val) )
9  print( len(val) )
```

types2.py

Other types we'll see

There are several other built-in types that we'll hear more about later like

`Boolean`: can have value True or False

`List`: for keeping track of a sequence of values

Other types we'll see

There are several other built-in types that we'll hear more about later like

`Boolean`: can have value True or False

`List`: for keeping track of a sequence of values

```
>>> type(False)
<class 'bool'>
>>> type([1,2,3])
<class 'list'>
```

Roadmap

- Expressions
- Data types
- Receiving input from user
  - Counterpart of showing an user some outputs
- Division operators

User Input

Python provides a function called `input` that gets input from the keyboard.

When this function is called, the program *stops and waits* for the user to type something.

When the user presses Return or Enter, the program resumes and input returns what the user typed as a string

User Input

Python provides a function called `input` that gets input from the keyboard.

When this function is called, the program *stops and waits* for the user to type something.

When the user presses Return or Enter, the program resumes and input returns what the user typed as a string

input_and_output.py

input_and_output.py

```
1  # author: Alimoor Reza
2  # description: receiving input from the user and displaying output
3
4  first_name = input("Please enter your first name")
5  last_name = input("Please enter your last name")
6  print("Hello ", first_name, " ", last_name)
```

User Input – Strings

By default, the value of variable that results from an input statement is a String.

Strings are very useful, but they are limited in that they only hold a sequence of characters.

- As such, you cannot perform mathematical operations on strings or string variables.
- if I want the result of the input function to give me a value that I can use in a calculation, I need to convert the string to a numeric format.

Let's build a temperature converter!

The formula for converting Fahrenheit temperatures to Celsius is

$$C = (F - 32) * \frac{5}{9}$$

We could make this an interactive program with code like this:

temparature.py

```
1  # Alimoor Reza
2  # Introduction to data types
3
4  # formula: C = (F-32)*(5/9)
5
6  fahrenheit_temp = input("Enter the temparature in fahrenheit scale:
7  celcius_temp = (fahrenheit_temp-32)*(5/9)
8  print("Converted temparature is: ", celcius_temp, " in Celcius")
```

Exercise #1:

Create a file called temperature.py

Type in the following code

$$C = (F - 32) * \frac{5}{9}$$

temparature.py

```
1  # Alimoor Reza
2  # Introduction to data types
3
4  # formula: C = (F-32)*(5/9)
5
6  fahrenheit_temp = input("Enter the temparature in fahrenheit scale:
7  celcius_temp = (fahrenheit_temp-32)*(5/9)
8  print("Converted temparature is: ", celcius_temp, " in Celcius")
```

Will it work? Why or why not??

Fix it.

Converting a string to an integer

Fortunately, there's a built-in function we can use to convert something to an integer: **`int()`**

temperature_converter_try2.py

temparature_converter_try2.py

```
1  # Alimoor Reza
2  # Introduction to data types
3
4  # formula: C = (F-32)*(5/9)
5
6  fahrenheit_temp_str = input("Enter the temparature in fahrenheit sca
7  fahrenheit_temp_int = int(fahrenheit_temp_str)
8  celcius_temp = (fahrenheit_temp_int-32)*(5/9)
9  print("Converted temparature is: ", celcius_temp, " in Celcius")
10
```

temperature_converter_try3.py

Or it can be done in one step like this

temparature_converter_try3.py

```
1  # Alimoor Reza
2  # Introduction to data types
3
4  # formula: C = (F-32)*(5/9)
5
6  fahrenheit_temp_int = int(input("Enter the temparature in fahrenheit
7  celcius_temp = (fahrenheit_temp_int-32)*(5/9)
8  print("Converted temparature is: ", celcius_temp, " in Celcius")
```

A tricky question

What would happen if the user typed a floating-point value like 72.5?

```
4  # formula: C = (F-32)*(5/9)
5
6  fahrenheit_temp_str = input("Enter the temparature in fahrenheit sca
7  fahrenheit_temp_int = int(fahrenheit_temp_str)
8  celcius_temp = (fahrenheit_temp_int-32)*(5/9)
9  print("Converted temparature is: ", celcius_temp, " in Celcius")
10
```

Other type conversion functions

Convert to a floating-point type with **`float()`**

Convert to a string with **`str()`**

temperature_converter_float.py

temparature_converter_float.py

```
1  # Alimoor Reza
2  # Introduction to data types
3
4  # formula: C = (F-32)*(5/9)
5
6  fahrenheit_temp_int = float(input("Enter the temparature in fahrenhe
7  celcius_temp = (fahrenheit_temp_int-32)*(5/9)
8  print("Converted temparature is: ", celcius_temp, " in Celcius")
9
10
11
```

Shell

```
>>> %Run temparature_converter_float.py

  Enter the temparature in fahrenheit scale: 65.5
  Converted temparature is:  18.611111111111111  in Celcius
```

Python 3.7.9

Other type conversion functions

Convert to a floating-point type with **`float()`**

Convert to a string with **`str()`**

user_input.py

user_input.py

```
1  # Alimoor Reza
2  # user input
3
4  target_number = 54
5  # the code below will break, any ideas why?
6  user_input = int( input("enter a number larger than ", target_number) )
```

user_input.py

user_input.py

```
1  # Alimoor Reza
2  # user input
3
4  target_number = 54
5  # the code below will break, any ideas why?
6  user_input = int( input("enter a number larger than ", target_number) )
```

Shell

```
>>> %Run user_input.py
  Traceback (most recent call last):
    File "/Users/reza/Class_and_Research/drake_university/drake_teaching/qingdao_university/cs65_fall25/cs65_codes/
  lecture3/user_input.py", line 6, in <module>
      user_input = int( input("enter a number larger than ", target_number) )
  TypeError: input expected at most 1 arguments, got 2

>>>
```

Previous code fixed

user_input_fixed.py

```
1  # Alimoor Reza
2  # user input fixed with str() function
3
4  target_number = 54
5  # the code below will break, any ideas why?
6  user_input = int( input("enter a number larger than " + str(target_number) ) )
```

Roadmap

- Expressions
- Data types
- Receiving input from user
  - Counterpart of showing an user some outputs
- Division operators

Division Operations

Floating-point division operator is simply this: /

The / operator will always give you a float as a result.

Division Operations

Floating-point division operator is simply this: /

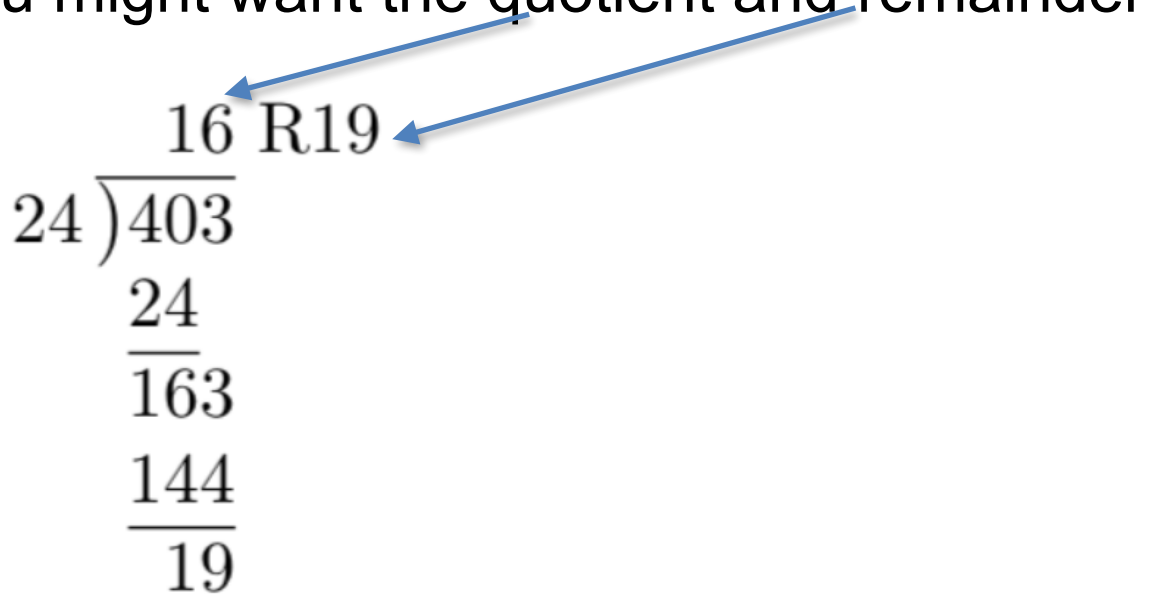
The / operator will always give you a float as a result.

```
>>> 320/20
16.0
>>> 403.0/24.0
16.791666666666668
```

```
>>> val = 4/2
>>> print(val)
 2.0
```

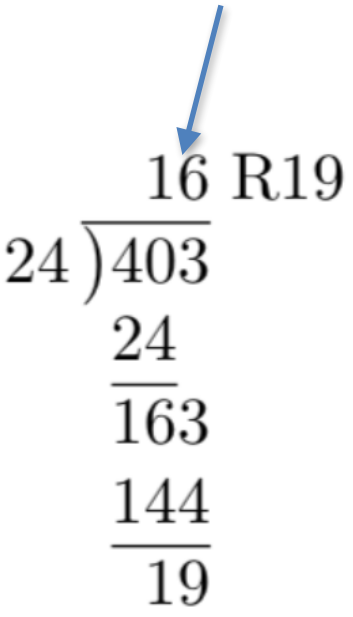
Integer Division Operations

Sometimes, instead of getting the full floating-point expansion, you might want the quotient and remainder separately.



Integer Division Operation //

The // is the floor division operator - it gives you the quotient and ignores and fractional/remainder part.




```
>>> 403//24
16
```

Integer Division Operator %

The % operator is the modulo operator - another word for remainder - it will ignore the quotient and just give you the remainder.

```
>>> 403%24
19
```

$$\begin{array}{r} 16\ \mathrm{R}19 \\ 24\overline{)403} \\ \underline{24} \\ 163 \\ \underline{144} \\ 19 \end{array}$$

Try it!

Try some of the following in your interactive shell. What pattern do you notice?

```
0 // 2
1 // 2
2 // 2
3 // 2
4 // 2
5 // 2

20 // 5
21 // 5
22 // 5
23 // 5
24 // 5
25 // 5
```

```
0 % 2
1 % 2
2 % 2
3 % 2
4 % 2
5 % 2

20 % 5
21 % 5
22 % 5
23 % 5
24 % 5
25 % 5
```

Good quiz questions…

What is the difference between // and /?

What is the difference between // and %?

When is this useful?

Modulo (%) is also good if you need to check if a number is even or odd

```
>>> 253 % 2 #odd numbers have a remainder of 1 when divided by 2

1
```

```
>>> 254 % 2 #even numbers have a remainder of 0 when divided by 2

0
```

When is this useful?

- Or, if you want to check if a number is divisible by 4 - say you're checking to see if the year is a presidential election year, Olympic year, leap year, etc., you can see if the result is 0

```
>>> 2024 % 4
0
```

When is this useful?

- There are 100 pieces of candy; there are 8 people at the party; how many pieces of candy does each person get? How much candy is left over?
- I'm programming a super computer with 64 processors to perform 1500 variations of a simulation for a physics experiment, and each of the simulations takes about the same amount of time to run. How many simulations should be given to each processor?
- I have 78 cents, how many quarters should I give in change?

Show: Change.py

Try it!

Write a program that will

- Prompt user for the number of slices of pizza;
- Prompt the user for the number of people

Print out how many slices each person can eat (if everyone gets the same number of slices)

Print out how many slices will be left over.

What to work on

Lab #1 due tonight on Tuesday, September 16th (sections#1 and #2) and Wednesday 17th (section#3)

Textbook Reading:

https://greenteapress.com/thinkpython2/html/thinkpython2002.html#sec10

Lab Activity

- To do: Finish the Lab1 and upload on Xuexitong (Fanya)

CS 65: Introduction to Computer Science (Fall 25)
Instructor: **Dr. Md Alimoor Reza**
Assistant Professor of Computer Science
Department of Mathematics and Computer Science
Drake University

Classroom: Boyi#508@Qingdao University
Meeting Time : Tuesday 14:00 pm - 15:50 pm (Week#1-Week#9), Thursday 10:10 am - 12:00 pm (Week#1-Week#17)
Office Hours: Tuesday: 08:00 am-9:50 am and and Thursday: 2:00 pm - 3:50 pm or by appointment

Course Syllabus: CS65 Fall'25
MOOC Portal: Chaoxing Fanya (Xue Xi Tong)

Section#1 Schedule
A tentative schedule is provided below (content may be adjusted as we progress).

Date	Topic	Reading	Items due
week 1 (Tue: Sep 09)	Introduction to Computer Science Lecture 1 slide **Lab 0 (Windows user)** **Lab 0 (Mac user)**	Reading: A Byte of Python (Why Python & its advantage)	
week 1 (Thu: Sep 11)	Displaying text Assignment statements Variables, expression Lecture 2 slide **Lab 1 (released on Sep 11)**	Reading: Chapter 2: Python for Everyone – Variables, expressions, and statements	
week 2 (Tue: Sep 16)	Expression Data Types Input Division Operator Lecture 3 slide	Reading: Python Values and Types	Lab 1 (due by Sep 16th for Sec#1 + Sec#2) Lab 1 (due by Sep 17th for Sec#3)

Lab Activity

- To do: Finish the book chapter reading

CS 65: Introduction to Computer Science (Fall 25)
Instructor: **Dr. Md Alimoor Reza**
Assistant Professor of Computer Science
Department of Mathematics and Computer Science
Drake University

Classroom: Boyi#508@Qingdao University
Meeting Time : Tuesday 14:00 pm - 15:50 pm (Week#1-Week#9), Thursday 10:10 am - 12:00 pm (Week#1-Week#17)
Office Hours: Tuesday: 08:00 am-9:50 am and and Thursday: 2:00 pm - 3:50 pm or by appointment

Course Syllabus: CS65 Fall'25
MOOC Portal: Chaoxing Fanya (Xue Xi Tong)

Section#1 Schedule
A tentative schedule is provided below (content may be adjusted as we progress).

Date	Topic	Reading	Items due
week 1 (Tue: Sep 09)	Introduction to Computer Science Lecture 1 slide **Lab 0 (Windows user)** **Lab 0 (Mac user)**	Reading: A Byte of Python (Why Python & its advantage)	
week 1 (Thu: Sep 11)	Displaying text Assignment statements Variables, expression Lecture 2 slide **Lab 1 (released on Sep 11)**	Reading: Chapter 2: Python for Everyone – Variables, expressions, and statements	
week 2 (Tue: Sep 16)	Expression Data Types Input Division Operator Lecture 3 slide	Reading: Python Values and Types	Lab 1 (due by Sep 16th for Sec#1 + Sec#2) Lab 1 (due by Sep 17th for Sec#3)