01 Prepare: Review Python

The concepts in CSE 111 build on the concepts that you learned in CSE 110. In order to be successful in CSE 111, it is important that you remember and understand the concepts from CSE 110. To help you remember those concepts, during lesson 1 of CSE 111, you will review programming concepts that you learned in CSE 110.

Concepts

Here is a list of the Python programming concepts and topics from CSE 110 that you should review during this lesson.

Comments

A **comment** in a computer program is a note or description that is supposed to help a programmer understand the program. Computers ignore comments in a program. In Python, a comment begins with the hash symbol (#) and extends to the end of the current line.

```
# This is a comment because it has
# hash symbols at the beginning.
```

Variables

A **variable** is a location in a computer's memory where a program stores a value. A variable has a name, a data type, and a value. In Python, we assign a value to a variable by using the assignment operator, which is the equals symbol (=). A computer may change the value and data type of a variable while executing a program.

```
length = 5
time = 7.2
in_flight = True
first_name = "Cho"
```

Data Types

Python has many **data types** including `str`, `bool`, `int`, `float`, `list`, and `dict`. Most of the data types that you will use in your programs in CSE 111 are shown in the following list.

- A **str** (string) is any text inside single or double quotes, any text that a user enters, and any text in a text file. For example:

  ```
  greeting = "Hello"
  text = "23"
  ```

- A **bool** (Boolean variable) is a variable that stores either `True` or `False`. A Boolean variable may not store any other value besides `True` or `False`. For example:

  ```
  found = True
  ```

- An **int** (integer) is a whole number like 14. An `int` may not have a fractional part or digits after the decimal point. For example:

  ```
  x = 14
  ```

- A **float** (floating point number) is a number that may have a fractional part or digits after the decimal point like 7.51. For example:

  ```
  sample = 7.51
  ```

- A **list** is a collection of values. Each value in a list is called an element and is stored at a unique index. The primary purpose of a list is to efficiently store many elements. In a Python program, we can create a list by using square brackets ([and]) and separating the elements with commas (,). For example here are two lists named `colors` and `samples`:

  ```
  colors = ["yellow", "red", "green", "yellow", "blue"]
  samples = [6.5, 7.2, 7.0, 8.1, 7.2, 6.8, 6.8]
  ```

  You will study lists in lesson 7 of this course.

- A **dict** (dictionary) is a collection of items. Each item is a key value pair. The primary purpose of a dictionary is to enable a computer to find items very quickly. In a Python program, we can create a dictionary by using curly braces ({ and }) and separating the items with commas (,). For example:

  ```
  students = {
      "42-039-4736": "Clint Huish",
      "61-315-0160": "Amelia Davis",
      "10-450-1203": "Ana Soares",
      "75-421-2310": "Abdul Ali",
      "07-103-5621": "Amelia Davis"
  }
  ```

  You will study dictionaries in lesson 8 of this course.

It is possible to convert between many of the data types. For example, to convert from any data type to a string, we use the `str()` function. To convert from a string to an integer, we use the `int()` function, and to convert from a string to a float, we use the `float()` function. The `int()` and `float()` functions are especially useful to convert user input, which is always a string, to a number. See the program in example 2 below.

User Input

In a Python program, we use the `input()` function to get input from a user in a terminal window. The `input` function always returns a string of characters.

```
text = input("Please enter your name: ")
color = input("What is your favorite color? ")
```

Displaying Results

In a Python program, we use the `print()` function to display results to a user. The easiest way to print both text and numbers together is to use a formatted string literal (also known as an f-string).

```python
print(f"Heart rate: {rate}")
```

The Python program in example 1 creates ten different variables. Some of the variables are of type `str`, some of type `bool`, some of type `int`, and some of type `float`. The program uses f-strings to print the name, data type, and value of each variable.

```python
# Example 1

# Create variables of different data types and then
# print the variable names, data types, and values.

a = "Her name is "  # string
b = "Isabella"      # string
c = a + b           # string plus string makes string
print(f"a: {type(a)} {a}")
print(f"b: {type(b)} {b}")
print(f"c: {type(c)} {c}")
print()

d = False   # boolean
e = True    # boolean
print(f"d: {type(d)} {d}")
print(f"e: {type(e)} {e}")
print()

f = 15      # int
g = 7.62    # float
h = f + g   # int plus float makes float
print(f"f: {type(f)} {f}")
print(f"g: {type(g)} {g}")
print(f"h: {type(h)} {h}")
print()

i = "True"   # string because of the surrounding quotes
j = "2.718"  # string because of the surrounding quotes
print(f"i: {type(i)} {i}")
print(f"j: {type(j)} {j}")
```

```
> python example_1.py
a: <class 'str'> Her name is
b: <class 'str'> Isabella
c: <class 'str'> Her name is Isabella

d: <class 'bool'> False
e: <class 'bool'> True

f: <class 'int'> 15
g: <class 'float'> 7.62
h: <class 'float'> 22.62

i: <class 'str'> True
j: <class 'str'> 2.718
```

The Python program in example 2 creates six different variables, some of type `string`, some of type `int`, and some of type `float`. Lines 4–5 and 7–8 of example 2 demonstrate that no matter what the user types, the `input()` function always returns a string. Lines 13 and 14 show how to use the `int()` and `float()` functions to convert a string to a number so that the numbers can be used in calculations.

```
 1  # Example 2
 2
 3  # The input function always returns a string.
 4  k = input("Please enter a number: ")        # string
 5  m = input("Please enter another number: ")  # string
 6  n = k + m           # string plus string makes string
 7  print(f"k: {type(k)} {k}")
 8  print(f"m: {type(m)} {m}")
 9  print(f"n: {type(n)} {n}")
10  print()
11
12  # The int and float functions convert a string to a number.
13  p = int(input("Please enter a number: "))          # int
14  q = float(input("Please enter another number: "))  # float
15  r = p + q                      # int plus float makes float
16  print(f"p: {type(p)} {p}")
17  print(f"q: {type(q)} {q}")
18  print(f"r: {type(r)} {r}")
```

```
> python example_2.py
Please enter a number: 6
Please enter another number: 4
k: <class 'str'> 6
m: <class 'str'> 4
n: <class 'str'> 64

Please enter a number: 5
Please enter another number: 3
p: <class 'int'> 5
q: <class 'float'> 3.0
r: <class 'float'> 8.0
```

Arithmetic

Python has many **arithmetic operators** including power (**), negation (-), multiplication (*), division (/), floor division (//), modulo (%), addition (+), and subtraction (-).

Operator Precedence

When we write an arithmetic expression that contains more than one operator, the computer executes the operators according to their **precedence**, also known as the **order of operations**. This table shows the precedence for the arithmetic operators.

Operators	Description	Precedence
( )	parentheses	highest
**	exponentiation (power)	↑
-	negation	|
* / // %	multiplication, division, floor division, modulo	|

Operators	Description	Precedence
+ -	addition, subtraction	↓
=	assignment	lowest

When an arithmetic expression includes two operators with the same precedence, the computer evaluates the operators from left to right. For example, in the arithmetic expression x / y * c the computer will first divide *x* by *y* and then multiply that result by *c*. If you need the computer to evaluate a lower precedence operator before a higher precedence one, you can add parentheses to the expression to change the evaluation order. The computer will always evaluate arithmetic that is inside parentheses first because parentheses have the highest precedence of all the arithmetic operators.

If this is the first time that you have encountered arithmetic operator precedence, you should watch this Khan Academy video: Introduction to Order of Operations (10 minutes).

The Python program in example 3 gets input from the user and converts the user input into two numbers on lines 9 and 10. Then at line 13 the program computes the length of a cable from the two numbers. Finally at line 17, the program uses an f-string to print the length rounded to two places after the decimal point.

```
 1  # Example 3
 2
 3  # Given the distance that a cable will span and the distance
 4  # it will sag or dip in the middle, this program computes the
 5  # length of the cable.
 6
 7  # Get user input and convert it from
 8  # strings to floating point numbers.
 9  span = float(input("Distance the cable must span in meters: "))
10  dip = float(input("Distance the cable will sag in meters: "))
11
12  # Use the numbers to compute the cable length.
13  length = span + (8 * dip**2) / (3 * span)
14
15  # Print the cable length in the
16  # console window for the user to see.
17  print(f"Length of cable in meters: {length:.2f}")
```

```
> python example_3.py
Distance the cable must span in meters: 500
Distance the cable will sag or dip in meters: 18.5
Length of cable in meters: 501.83
```

In example 3, the arithmetic that is written on line 13 comes from a well known formula. Given the distance that a cable must span and the vertical distance that the cable will be allowed to sag or dip in the middle of the cable, the formula for calculating the length of the cable is:

$$length = span + \frac{8\,dip^2}{3\,span}$$

Shorthand Operators

The Python programming language includes many **augmented assignment operators**, also known as **shorthand operators**. All the shorthand operators have the same precedence as the assignment operator (=). Here is a list of some of the Python shorthand operators:

```
**=   *=   /=   //=   %=   +=   -=
```

To understand what the shorthand operators do and why Python includes them, imagine a program that computes the price of a pizza. The price of a large pizza with cheese and no other toppings is $10.95. The price of each topping, such as ham, pepperoni, olives, and pineapple is $1.45. Here is a short example program that asks the user for the number of toppings and computes the price of a pizza:

```
 1  # Example 4
 2
 3  # Compute the total price of a pizza.
 4
 5  # The base price of a large pizza is $10.95
 6  price = 10.95
 7
 8  # Ask the user for the number of toppings.
 9  number_of_toppings = int(input("How many toppings? "))
10
11  # Compute the cost of the toppings.
12  price_per_topping = 1.45
13  toppings_cost = number_of_toppings * price_per_topping
14
15  # Add the cost of the toppings to the price of the pizza.
16  price = price + toppings_cost
17
18  # Print the price for the user to see.
19  print(f"Price: ${price:.2f}")
```

```
> python example_4.py
How many toppings? 3
Price: $15.30
```

The statement at line 16 in example 4 causes the computer to get the value in the *price* variable which is 10.95, then add the cost of the toppings to 10.95, and then store the sum back into the *price* variable. Python includes a shorthand operator that combines addition (+) and assignment (=) into one operator (+=). We can use this shorthand operator to rewrite line 16 like this:

```
price += toppings_cost
```

This statement with the shorthand operator is equivalent to the statement on line 16 of example 4, meaning the two statements cause the computer to do the same thing. Example 5 contains the same program as example 4 but uses the shorthand operator += at line 16.

```
 1  # Example 5
 2
 3  # Compute the total price of a pizza.
 4
 5  # The base price of a large pizza is $10.95
```

```python
price = 10.95

# Ask the user for the number of toppings.
number_of_toppings = int(input("How many toppings? "))

# Compute the cost of the toppings.
price_per_topping = 1.45
toppings_cost = number_of_toppings * price_per_topping

# Add the cost of the toppings to the price of the pizza.
price += toppings_cost

# Print the price for the user to see.
print(f"Price: ${price:.2f}")
```

```
> python example_5.py
How many toppings? 3
Price: $15.30
```

if Statements

In Python, we use `if` statements to cause the computer to make decisions; `if` statements are also called **selection** statements because the computer selects one group of statements to execute and skips the other group of statements.

There are six comparison operators that we can use in an `if` statement:

<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equal to
!=	not equal to

Example 6 contains Python code that checks if a number is greater than 500.

```python
# Example 6

# Get an account balance as a number from the user.
balance = float(input("Enter the account balance: "))

# If the balance is greater than 500, then
# compute and add interest to the balance.
if balance > 500:
    interest = balance * 0.03
    balance += interest

# Print the balance.
print(f"balance: {balance:.2f}")
```

```
> python example_6.py
Enter the account balance: 350
balance: 350.0

> python example_6.py
Enter the account balance: 525
balance: 540.75
```

If you have written programs in other programming languages such as JavaScript, Java, or C++, you always used curly braces to mark the start and end of the body of an `if` statement. However, notice in example 6 that `if` statements in Python do not use curly braces. Instead, we type a colon (:) after the comparison of the `if` statement as shown on line 8. Then we indent all the statements that are in the body of the `if` statement as shown on lines 9 and 10. The body of the `if` statement ends with the first line of code that is not indented, like line 13.

It may seem strange to not use curly braces to mark the start and end of the body of an `if` statement. However, the Python way forces us to write code where the indentation matches the functionality or in other words, the way we indent the code matches the way that the computer will execute the code.

if … elif … else Statements

Each `if` statement may have an `else` statement as shown in example 7 on line 13. We can combine `else` and `if` into the keyword `elif` as shown on lines 9 and 11.

```
 1  # Example 7
 2
 3  # Get the cost of an item from the user.
 4  cost = float(input("Please enter the cost: "))
 5
 6  # Determine a discount rate based on the cost.
 7  if cost < 100:
 8      rate = 0.10
 9  elif cost < 250:
10      rate = 0.15
11  elif cost < 400:
12      rate = 0.18
13  else:
14      rate = 0.20
15
16  # Compute the discount amount
17  # and the discounted cost.
18  discount = cost * rate
19  cost -= discount
20
21  # Print the discounted cost for the user to see.
22  print(f"After the discount, you will pay {cost:.2f}")
```

```
> python example_7.py
Please enter the cost: 300
After the discount, you will pay 246.0
```

Logical Operators

Python includes two **logic operators** which are the keywords `and`, `or` that we can use to combine two comparisons. Python also includes the logical `not` operator. Notice in Python that the logical operators are literally the words: `and`, `or`, `not` and not symbols as in other programming languages:

```
if driver >= 54 or (driver >= 32 and passenger >= 54):
    message = "Enjoy the ride!"
```

Videos

If any of the concepts or topics in the previous list seem unfamiliar to you, you should review them. To review the unfamiliar concepts, you could rewatch some of the Microsoft videos about Python that you watched for CSE 110:

Input and print Functions (4 minutes)

Demonstration of print Function (6 minutes)

Comments (3 minutes)

String Data Type (5 minutes)

Numeric Data Types (6 minutes)

Conditional Logic (5 minutes)

Handling Multiple Conditions (6 minutes)

Complex Conditions (4 minutes)

Tutorials

Reading these tutorials may help you recall programming concepts from CSE 110.

Why Choose Python?

Interacting with Python

Basic Data Types in Python

Variables in Python

Operators and Expressions in Python

Conditional Statements in Python

You could also read some of the Python tutorials at W3 Schools. Or you could search for "Python" in the BYU-Idaho library online catalog and read one of the online books from the result set.

Summary

During this lesson, you are reviewing the programming concepts that you learned in CSE 110. These concepts include how to do the following:

- write a comment
- use the `input` and `print` functions
- use the `int` and `float` functions to convert a value from a string to a number
- store a value in a variable

- perform arithmetic
- write `if … elif … else` statements
- use the logical operators `and`, `or`, `not`