Computational Thinking with Programming

python™

Lecture - 3

Data and Expressions: Literals, Variables and Identifiers

@cse_bennett @csebennett

BENNETT UNIVERSITY
TIMES OF INDIA GROUP

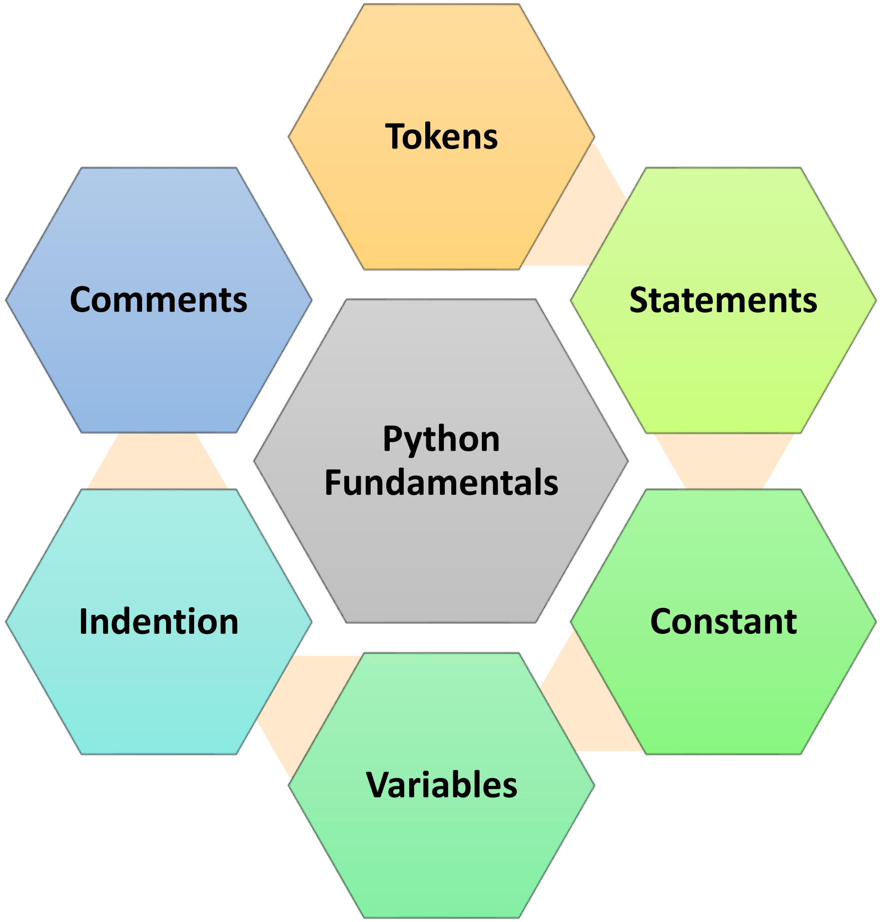
Today's Outline

- Previous Session:
  - Introduction to Basics of computer science, Hardware and Software.
  - Computational problem solving and introduction to python.

- Today's Session:
  - Basic Elements of Python Programs
    - Literals, Assignments.
    - Datatype Conversion.
    - Identifiers, and Expressions.

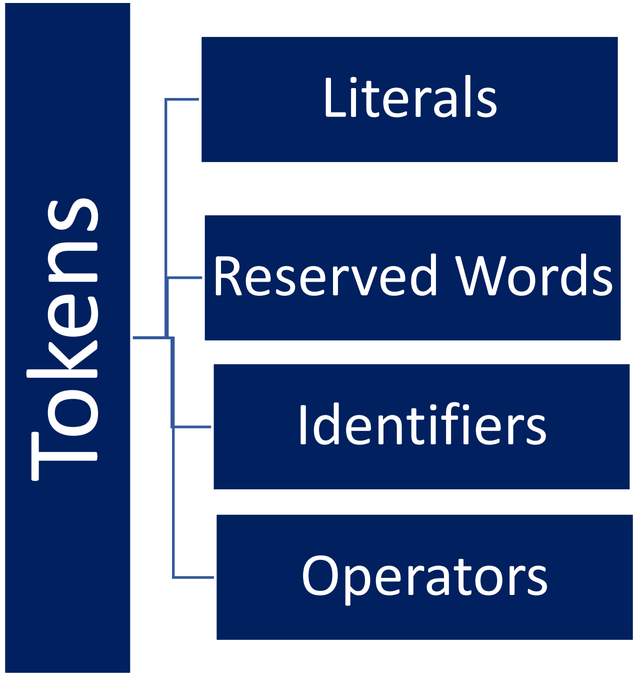
- Hands on Session with Jupyter Notebook:
  - We will practice on the Python basic elements in Jupyter Notebook.

Fundamentals of Python



Tokens

Tokens are the smallest unit of the program.



Tokens

Literals

Reserved Words

Identifiers

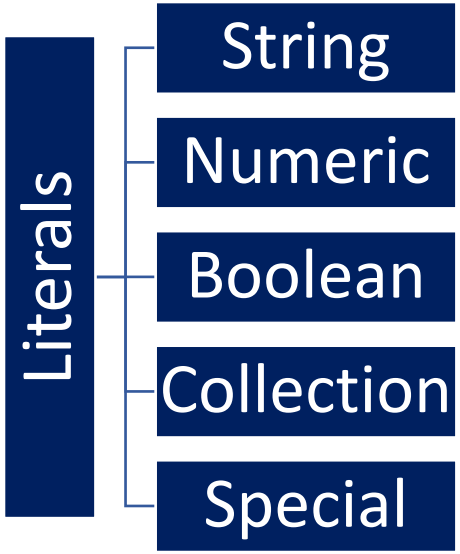
Operators

Literals

- In the following example, the parameter values passed to the print function are all technically called *literals*
    - More precisely, “Bennett University” and “Welcome to Bennett” are called *textual literals*, while 3 and 2.3 are called *numeric literals*

```
>>> print(“Bennett University")
Bennett University
>>> print(“Welcome to Bennett")
Welcome to  Bennett
>>> print(3)
3
>>> print(2.3)
2.3
```

Literals

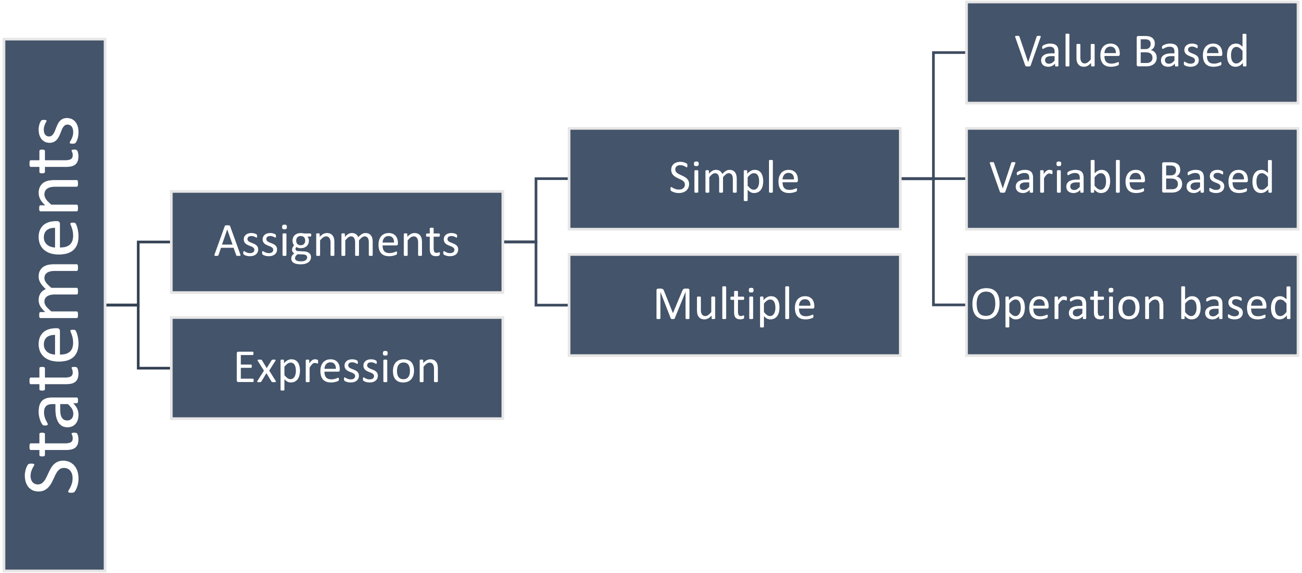


- String Literals:
    - A string literal is a sequence of characters surrounded by quotes. We can use both single, double or triple quotes for a string. And, a character literal is a single character surrounded by single or double quotes.
- Numeric Literals:
    - Numeric Literals are immutable (unchangeable). Numeric literals can belong to 3 different numerical types Integer, Float, and Complex.
- Boolean Literals:
    - A Boolean literal can have any of the two values: True or False.
- Collection literals:
    - There are four different literal collections List literals, Tuple literals, Dict literals, and Set literals.
- Special literals:
    - Python contains one special literal i.e. None. We use it to specify to that field that is not created.

Statements

- Python statements are nothing but logical instructions that interpreter can read and execute. It can be both single and multiline.

There are two categories of statements in Python:



Simple Assignment Statements

- A literal is used to indicate a specific value, which can be *assigned* to a *variable*
    - x is a variable and 2 is its value

```
>>> x = 2
>>> print(x)
2
>>> x = 2.3
>>> print(x)
2.3
```

Simple Assignment Statements

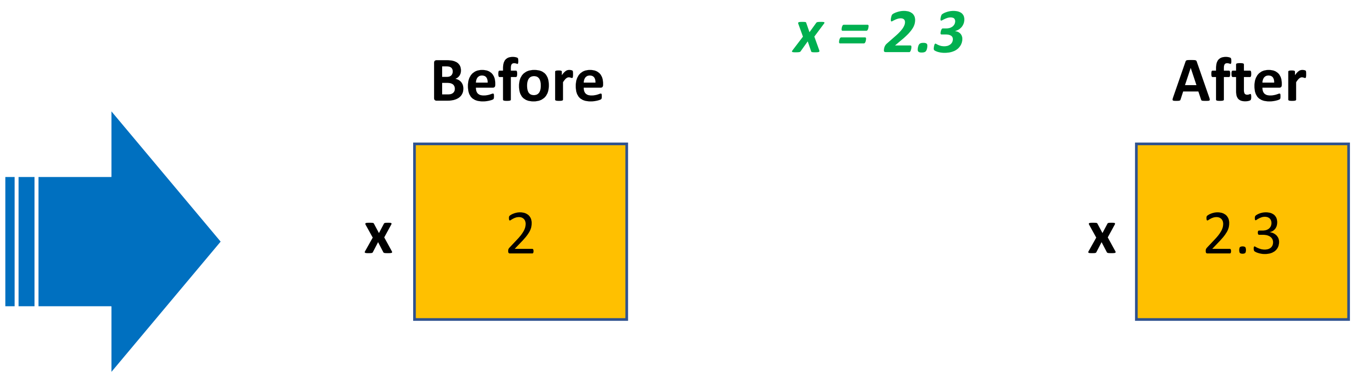
- A literal is used to indicate a specific value, which can be *assigned* to a *variable*
  - x is a variable and 2 is its value
  - x can be assigned different values; hence, it is called a variable

```
>>> x = 2
>>> print(x)
2
>>> x = 2.3
>>> print(x)
2.3
```

Simple Assignment Statements: What we Think

- A simple way to view the effect of an assignment is to assume that when a variable changes, its old value is replaced

```
>>> x = 2
>>> print(x)
2
>>> x = 2.3
>>> print(x)
2.3
```

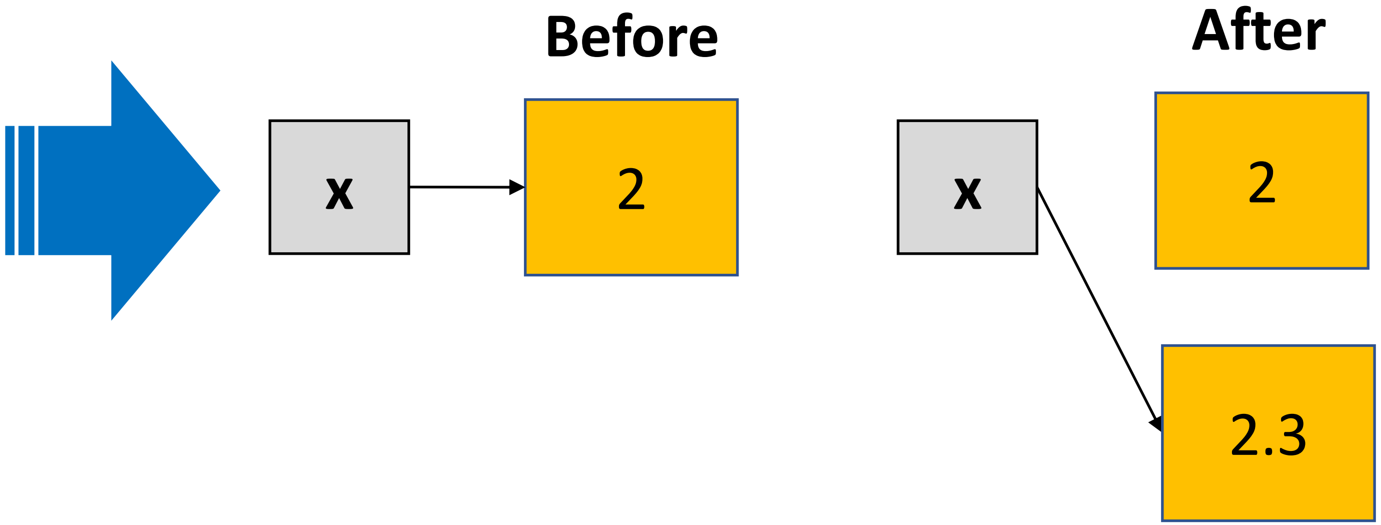


Simple Assignment Statements: What Actually Happen

- Python assignment statements are slightly different from the “variable as a box” model
  - In Python, values may end up anywhere in memory, and variables are used to refer to them

```
>>> x = 2
>>> print(x)
2
>>> x = 2.3
>>> print(x)
2.3
```

x = 2.3

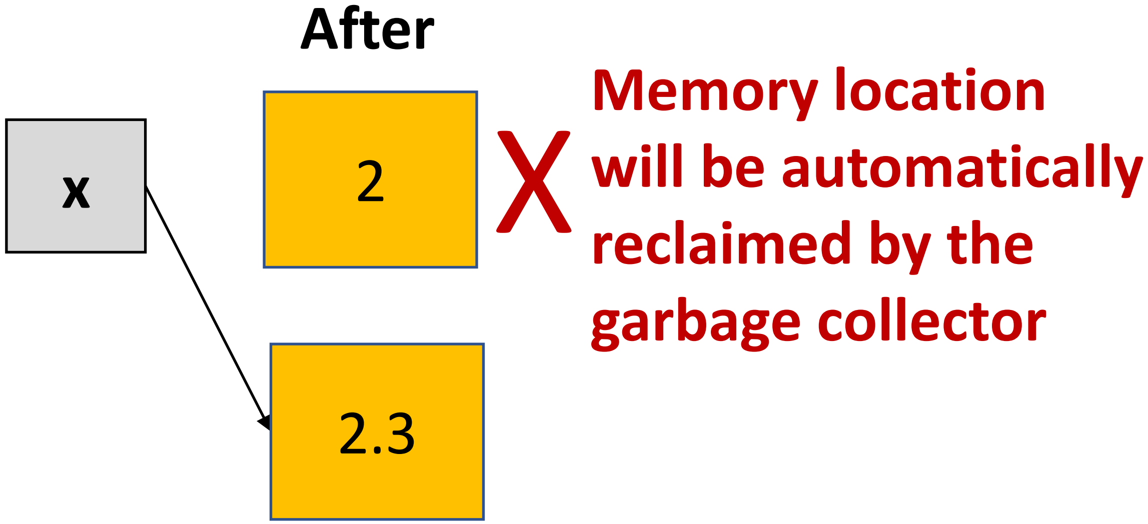


What will happen to value 2?

Garbage Collection

- Interestingly, as a Python programmer you do not have to worry about computer memory getting filled up with old values when new values are assigned to variables

- Python will automatically clear old values out of memory in a process known as *garbage collection*


After
x
2
X
2.3
Memory location will be automatically reclaimed by the garbage collector

Simultaneous Assignment

- Python allows us also to assign multiple values to multiple variables all at the same time

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
>>>
```

- This form of assignment might seem strange at first, but it can prove remarkably useful (e.g., for swapping values)

Simultaneous Assignment

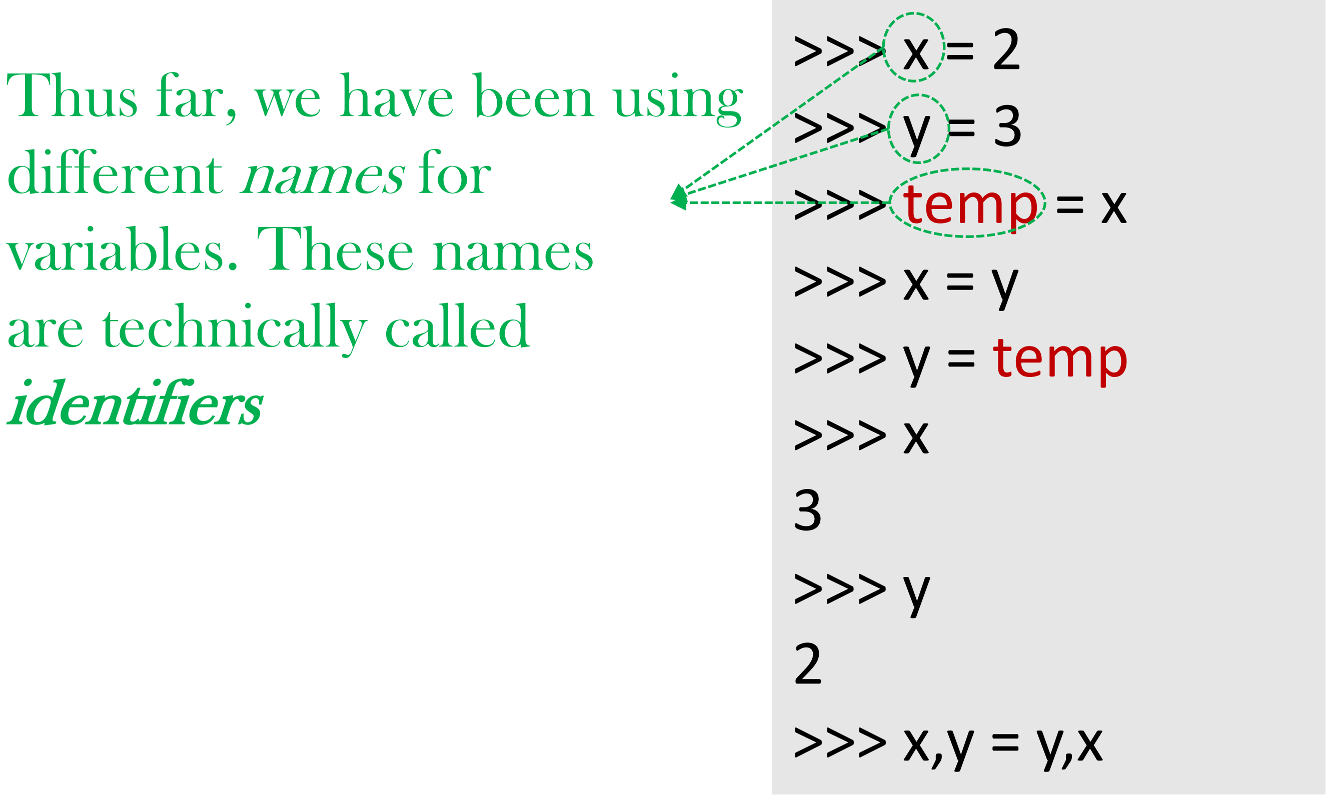
- Suppose you have two variables x and y, and you want to swap their values (*i.e., you want the value stored in x to be in y and vice versa*)

```
>>> x = 2
>>> y = 3
>>> x = y
>>> y = x
>>> x
3
>>> y
3
```

X **CANNOT be done with *two* simple assignments**

Simultaneous Assignment

- Suppose you have two variables x and y, and you want to swap their values (*i.e., you want the value stored in x to be in y and vice versa*)



Thus far, we have been using different *names* for variables. These names are technically called ***identifiers***

```
>>> x = 2
>>> y = 3
>>> temp = x
>>> x = y
>>> y = temp
>>> x
3
>>> y
2
>>> x,y = y,x
```

✓ **CAN be done with *three* simple assignments, but more efficiently with simultaneous assignment**

Identifiers

- Python has some rules about how identifiers can be formed
  - Every identifier must begin with a letter or underscore, which may be followed by any sequence of letters, digits, or underscores

```
>>> x1 = 10
>>> x2 = 20
>>> y_effect = 1.5
>>> celsius = 32
>>> 2celsius
  File "<stdin>", line 1
    2celsius
           ^
SyntaxError: invalid
syntax
```

Identifiers

- Python has some rules about how identifiers can be formed
  - Identifiers are *case-sensitive*

```
>>> x = 10
>>> X = 5.7
>>> print(x)
10
>>> print(X)
5.7
```

Identifiers

- Python has some rules about how identifiers can be formed
  - Some identifiers are part of Python itself (they are called *reserved words* or *keywords*) and cannot be used by programmers as ordinary identifiers

False
None
True
and
as
assert
break
class
continue
def
del
elif
else
except
finally
for
from
global
if
import
in
is
lambda
nonlocal
not
or
pass
raise
return
try
while
with
yield

Python Keywords

Identifiers

- Python has some rules about how identifiers can be formed
    - Some identifiers are part of Python itself (they are called *reserved words* or *keywords*) and cannot be used by programmers as ordinary identifiers

An example…

```
>>> for = 4
  File "<stdin>", line 1
    for = 4
        ^
SyntaxError: invalid syntax
```

Expressions

- You can produce new data (numeric or text) values in your program using *expressions*
  - This is an expression that uses the *addition operator*

```
>>> x = 2 + 3
>>> print(x)
5
>>> print(5 * 7)
35
>>> print("5" + "7")
57
```

Expressions

- You can produce new data (numeric or text) values in your program using *expressions*
  - This is an expression that uses the *addition operator*
  - This is another expression that uses the *multiplication operator*

```
>>> x = 2 + 3
>>> print(x)
5
>>> print(5 * 7)
35
>>> print("5" + "7")
57
```

Expressions

- You can produce new data (numeric or text) values in your program using *expressions*
  - This is an expression that uses the *addition operator*
  - This is another expression that uses the *multiplication operator*
  - This is yet another expression that uses the *addition operator* but to *concatenate* (or glue) strings together

```
>>> x = 2 + 3
>>> print(x)
5
>>> print(5 * 7)
35
>>> print("5" + "7")
57
```

Expressions

- You can produce new data (numeric or text) values in your program using *expressions*

Another example…

```
>>> x = 6
>>> y = 2
>>> print(x - y)
4
>>> print(x/y)
3.0
>>> print(x//y)
3
```

Yet another example…

```
>>> print(x*y)
12
>>> print(x**y)
36
>>> print(x%y)
0
>>> print(abs(-x))
6
```

Expressions: Summary of Operators

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Float Division
**	Exponentiation
abs()	Absolute Value
//	Integer Division
%	Remainder

Python Built-In Numeric Operations

Script

- One problem with entering code interactively into a Python shell is that the definitions are lost when we quit the shell
  - If we want to use these definitions again, we have to type them all over again!

- To this end, programs are usually created by typing definitions into a separate file called a *module* or *script*
  - This file is saved on disk so that it can be used over and over again

- A Python module file is just a text file with a *.py extension*, which can be created using any program for editing text (e.g., notepad or vim)

Programming Environments and IDLE

- A special type of software known as a *programming environment* simplifies the process of creating modules/programs.

- A programming environment helps programmers write programs and includes features such as automatic indenting, color highlighting, and interactive development.

- The standard Python distribution includes a programming environment called IDLE that you can use for working on the programs of this course.

Summary

- Programs are composed of statements that are built from *identifiers* and *expressions*

- Identifiers are names
  - They begin with an underscore or letter which can be followed by a combination of letter, digit, and/or underscore characters
  - They are case sensitive

- Expressions are the fragments of a program that produce data
  - They can be composed of *literals*, *variables*, and *operators*

Summary

- A literal is a representation of a specific value (e.g., 3 is a literal representing the number three)

- A variable is an identifier that stores a value, which can change (hence, the name *variable*)

- Operators are used to form and combine expressions into more complex expressions (e.g., the expression x + 3 * y combines two expressions together using the + and * operators)

Summary

- In Python, *assignment* of a value to a variable is done using the equal sign (i.e., =)
- Using assignments, programs can get inputs from users and manipulate them internally
- Python allows *simultaneous assignments*, which are useful for swapping values of variables

Thank You

?