Lecture 2
Variables & Expressions

Summer 2026

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Announcements

- Homework 0 is now available on the course website!
  - Due this Friday by 11:59pm
- First Section is tomorrow, Thursday June 25th
  - Written Check-in 1 will start there
- Make sure you have access to all the course resources
  - Gradescope – submit assignments and see grades/feedback
  - Ed – discussion board/Q&A/announcments
  - JupyterHub – coding environment

What makes up a Python program?

- Python code is composed of (mostly) expressions, statements, variables and methods/functions.
- An **expression** evaluates to a value
- A **statement** causes an effect (does not always evaluate to a value)
- Expressions and statements can contain variables and use methods/functions
- When put together, you create a **Python program** 

The **Python interpreter** will help us play around with evaluating expressions and using statements

Examples

- **Expressions**

```
3 + 4

pi * r ** 2
```

- **Statements**

```
pi = 3.14159

print(pi)
```

Assigns the **value** of 3.14159 to the **variable pi**

Prints the **value** in the **variable pi**

- **Expressions and Statements**

```
(pi * 3) + (r * 2)

print(pi * r ** 2)

3 + print(pi)
```

ERROR! A statement (e.g. print) cannot appear in the expression.

What does it mean to add the value of 3 to something being printed?

Expressions in Python can get complicated

- Python expressions can really messy, really quickly depending on how you write them.

Easy to evaluate

```
3 + 5
8 / 2
```

A bit harder

```
2**3/2*3-4
72-32/9.0*5
```

Use parentheses to control evaluation order

```
(2**3)/(2*3-4)
```

```
(72-32)/(9.0*5)
```

Not getting the expected value?

Use parentheses to get the order you want.

Order of Operations in Python

Operator	Description	Precedence	Direction
( )	Parenthesis	1 (Highest Precedence)	
**	Exponentiation	2	
//, /, *, %	Integer division, Float division, multiplication, and modulus (i.e. remainder) (same level of precedence)	3	Left to right
+, -	Addition and Subtraction (same level of precedence)	4 (Lowest Precedence)	Left to right

How is an expression evaluated?

(72 – 32) / (9.0 * 5)

(**72 – 32**) / (**9.0 * 5**)
(**40**) / (**9.0 * 5**)
40 / (**9.0 * 5**)
40 / (**45.0**)
40 / 45.0
.888

Another evaluation example

(72 – 32) / 9.0 * 5

(**72** – **32**) / **9.0** * **5**
(**40**) / **9.0** * **5**
40 / **9.0** * **5**
4.44 * **5**
22.2

Variables hold values

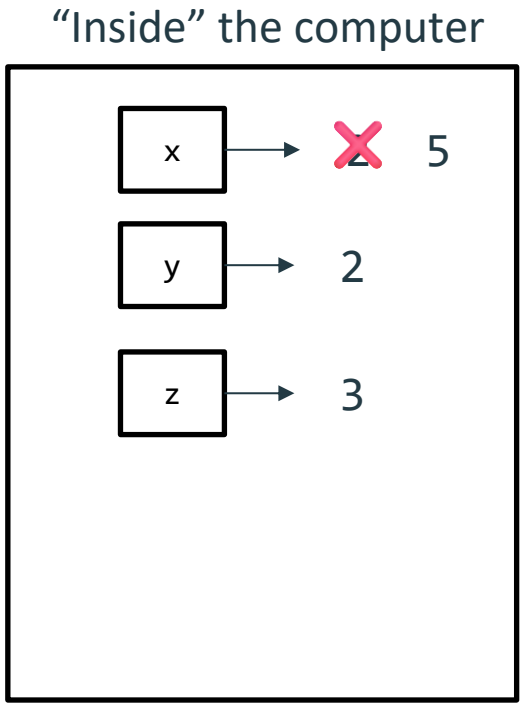
- Variables are “containers” that store values
  - Recall algebra (e.g. x = 5)
- Variables take the form of *`variable_name = expression`*
  - `x = 5`
  - `y = 3 + 5`
  - `z = x + y`

1. Evaluate the right hand side to a value
2. Store the value into the variable

- Not all variable names are permitted!
  - Must start with a character (e.g. 5 = 5 is not allowed, but five = 5 is!)
  - Special keywords (e.g. `True/False, lambda, import,`…)
- Variables have naming conventions/“rules”
  - More information in the Code Quality Guide

Changing Existing Variables

```
x = 2
print(x)
y = x
print(y)
z = x + 1
print(z)
x = 5
print(x)
print (y)
print(z)
```



Printed output

2
2
3
5
2
3

Data Types in Python

- **Integers (int)**
  - Whole numbers (including negatives)
  - `1, 2, 3, -1, -100, 1000`
- **Real numbers (float)**
  - float, for “floating point”
  - Includes decimals; `3.14, 2.718, 1.9999`
- **Strings (str)**
  - Represent words, characters, and sentences
  - “`Hello`”, “`Python`”, “`a`”, “`I love Python!`”
- **Booleans (bool)**
  - `True` or `False`
  - Represents truth values
  - Special operators: <, <=, >, >=, ==, not, and, or

Not sure what the type of a value or variable is in your code?

Use the built-in **`type()`** method

```
type(5) → int
type(“hi”) → str
type(True) → bool
```

Boolean Truth Tables

A	B	A and B	A or B	A ^ B
True	True	True	True	False
True	False	False	True	True
False	True	False	True	True
False	False	False	False	False

A	not A
True	False
False	True

Exclusive or (XOR) which is denoted by the **‘^’** character will not be used in this class

Data Types determine which operations you can perform

What will each expression evaluate to? Which will cause an error?

(2 ** 3) / 8	5 / 2	True * 2
11 % 2	5 // 2	False + (True – 1)
“Hello” * 3	“CSE” + “160”	True and (False and False)
“World” - “rld”	3 + “4”	not(True or False)
True or False	3 + True	True or False or not(True)
not(100 >= 100)	3 - False	

Put it all together to do something useful

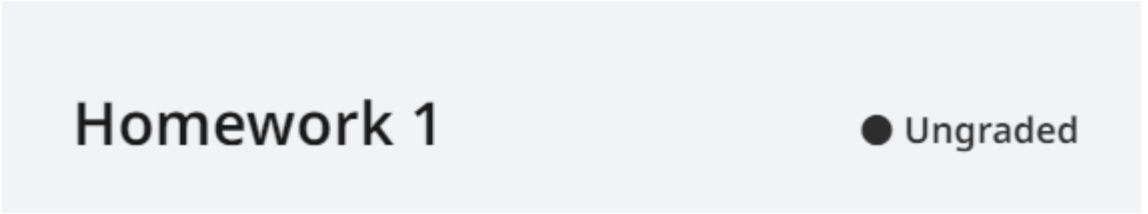
```
x = 1
y = 2
x + y
print(x + y)
print("The sum of", x, "and", y, "is", x + y)
```

- A program is a sequence of instructions performed in a specified order
- The computer executes each instruction one after another
  - An expression is evaluated
  - A statement changes the state of something (e.g. a variable update or prints something out)
- Expressions and statements bundled together that can be saved and reused by you or others.

Additional Resources

Using Gradescope for CSE160

Ungraded



You will see some, but not all of the autograder tests

Can resubmit as many times until the due date

Graded

Automatically Graded

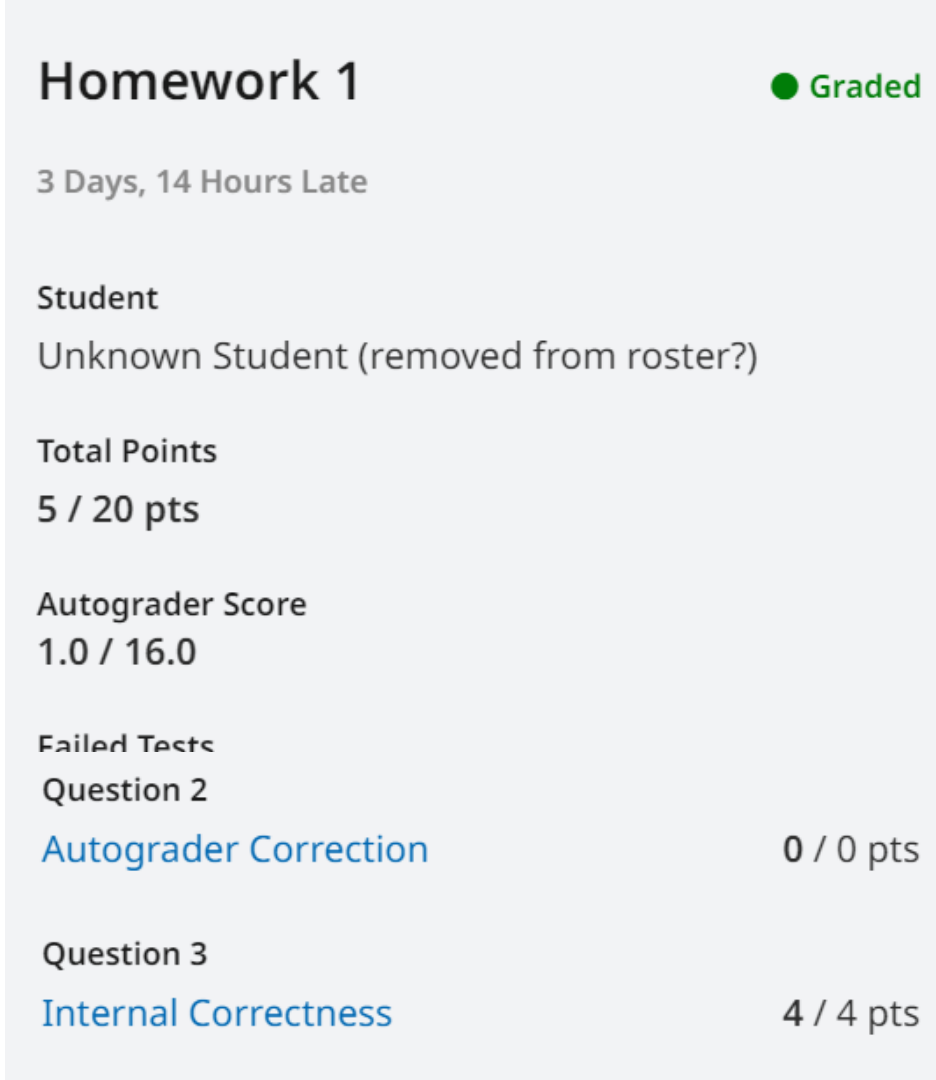
Autograder Score: Points given from all autograder tests

Manually Graded

Autograder Correction: TA's giving/taking away points in case of autograder error

Internal Correctness: Points given for following directions, good style, efficiency, etc.

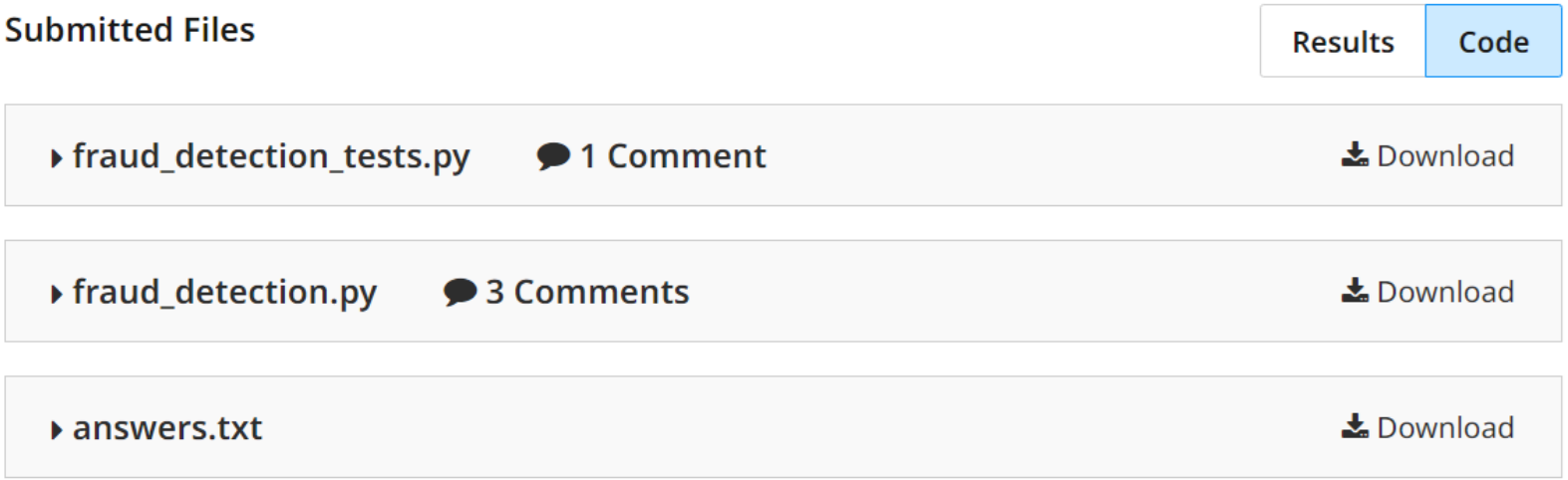
Autograder + Autograder Correction + Internal Correctness = Total Score



Comments

Autograder Results

Two tabs - results and code. Click on ‘Code’ to see list of files you submitted. Each file will have the number of comments that were made on it.



Comments Example

fraud_detection_tests.py 1 Comment Download

```
1  import fraud_detection as fd
2  import math
3
4
5  def test_ones_and_tens_digit_histogram():
```

Instructor | 03/12 at 12:23 pm

> -0: Small note here, it would also be good practice to include docstrings for test functions as well.

```
6      # Easy to calculate case: 5 numbers, clean percentages.
```