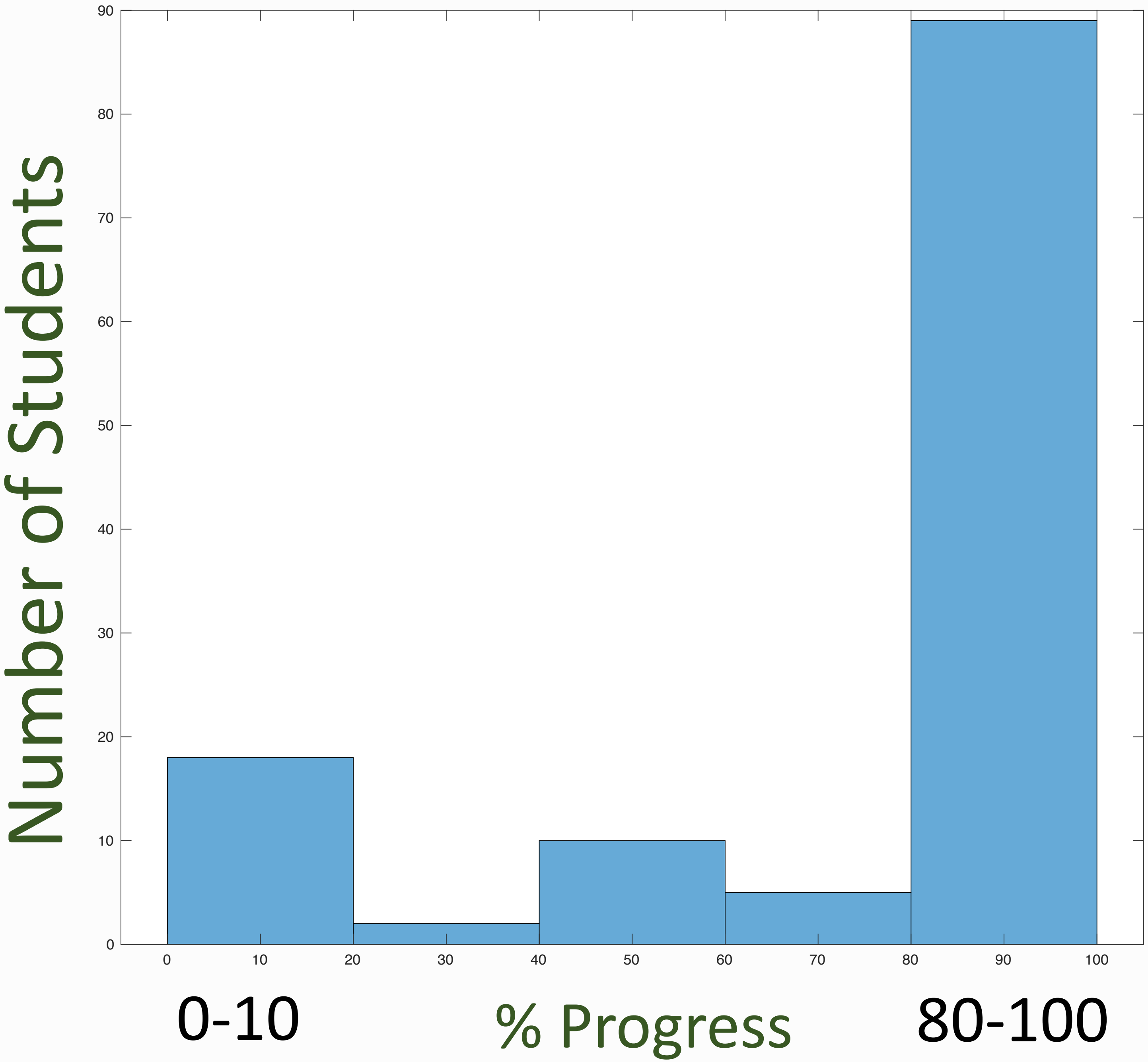
Current Assignments

- Homework 1 is due on Wednesday
- Homework 2 is due on Sunday, Feb 2nd

- Quiz 1 is this week in your lab sections


Number of Students
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100
0-10
% Progress
80-100

iClickers

- Raise you hand if you bought an iClicker specifically for this class.

Variables, Expressions, and Types

Prof Matthew Fricke

Version 1.0. Send corrections to mfricke@unm.edu.

Objects and Operators

- We can loosely divide computation into data and operations on that data.
- Objects contain data.
- The Type of an object defines the operations that can be performed on it.

- Everything in Python is an object so it is called an object oriented language.

- We will learn a lot more about how objects work later in the course.

Objects and Operators

```
>>> 6
6
```

Entering 6 into the python interpreter creates an object of type integer that contains the value 6

Python helpfully shows the value of the object just created.

Objects and Operators

```
>>> “some text”
‘some text’
```

Entering “some text” into the python interpreter creates an object of type string that contains the value ‘some text’

Python helpfully shows the value of the object just created.

Objects and Operators

```
>>> 6 + 5
11
```

Entering 6 + 5 creates two integer objects with the values 6 and 5, and give them to the operator +.

Python helpfully shows the result of applying the operator to the objects: 11.

Expressions

```
>>> 6 + 5
11

>>> 6
6

>>> “some text”
‘some text’
```

Creating objects and combining them with operators are expressions.

We will see other kinds of code that are expressions later.

Types, Objects and Operators

```
>>> 6 + 5
11
```

Objects and operators come together.

Only some operators work on particular objects.

For example the + operator is defined for integers

Types, Objects and Operators

```
>>> 12/2
6.0
```

... as is the division, /, operator.

```
>>> "some text"/"some other text"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

But the division operator is not defined for objects of type string.

Types, Objects and Operators

```
>>> 12/2
6.0

>>> "some text"/"some other text"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Python helpfully prints an error message telling you that / is not defined for objects of type string.

Types, Objects and Operators

Python provides several built-in object types:

Numeric Types:

Integers (int) –whole numbers, e.g. -1, 15, 42

Floating point (float) – fractions, e.g. 12.8, 0.6, -0.2

Complex (complex) – numbers with a real and imaginary parts, e.g. 3+7j

Boolean (bool): True and false values, e.g. True and False

Python 2 also had long ints but Python 3 ints can hold any size number

Types, Objects and Operators

```
>>> 4.3*3
12.899999999999999
>>> 5+3j + 2-7j
(7-4j)
>>> True and False
False
>>> True or False
True
>>>
```

Types, Objects and Operators

```
>>> "some text" + "some other text"
'some textsome other text’
>>> "some text"*4
'some textsome textsome textsome text'
```

Python defines lots of operators that might not do what you expect.

Types, Objects and Operators

```
>>> 78/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

Python will try to save you from common errors.

Types, Objects and Operators

```
>>> + +
  File "<stdin>", line 1
    + +
      ^
SyntaxError: invalid syntax
```

Syntax errors occur when the source code you enter is not understood by python.

Here we tried to apply an operator to another operator instead of an object.

Types, Objects and Operators

```
>>> (-1)**(1/2)
(6.123233995736766e-17+1j)
```

6.123233995736766e-17 is ALMOST zero

```
>>> 2/3
0.6666666666666666
```

An operator takes two objects and returns a result.

As we saw with multiplication of strings by integers the two input objects don’t have to have the same type.

The resulting object returned may or may not have the same type.

** is the exponentiation operator

Types, Objects and Operators

```
>>> 2/3
0.6666666666666666
>>> 2//3
0
>>> 10//3
3
>>> 10/3
3.3333333333333335
>>> 3/1
3.0
```

For example there are two division operators for integer objects.

// and /

// always returns an integer

/ always returns a float

Types, Objects and Operators

```
>>> 10%10
0
>>> 10%9
1
>>> 10%8
2
>>> 10%7
3
>>> 10%6
4

>>> 2%3
2
```

```
>>> 10%5
0
>>> 10%4
2
>>> 10%3
1
>>> 10%2
0
>>> 10%1
0

>>> 149%50
49
```

Modulus

The % operator return the remainder after division.

Types, Objects and Operators

```
>>> 10 > 3
True
>>> 10 < 3
False
>>> 10 == 10
True
>>> 10 != 3
True
>>> 10 >= 3
True
>>> 10 <= 3
False
```

```
>>> 10.000000000000001 == 10.000000000000003
False

>>> abs(10.000000000000001 - 10.000000000000003) < 0.00001
True
```

Comparison Operators

Always return an object of type bool.

We have to be careful when checking if floats are equal. Floats are often approximations (recall 6.123233995736766e-17 instead of zero).

Types, Objects and Operators

Sequence Types

Sequences consist of many values together.

String – a sequence of characters
List – a sequence of values that you can change (it's mutable)
Tuple – a sequence of value that cannot be changed (immutable)
Range – a sequence of integers

Types, Objects and Operators

```
>>> "This is a sequence of 35 characters"
'This is a sequence of 35 characters’

>>> ["this", "list", "has", 5, "elements"]
['this', 'list', 'has', 5, 'elements’]

>>> ('this', 'tuple', 'has', 5, 'elements')
('this', 'tuple', 'has', 5, 'elements')

>>> range(6)
range(0, 6)

>>> range(6,10,2)
range(6, 10, 2)
```

Range objects are a sequence of integers

The first number is the start integer, the second Integer is the last integer, and the last integer is the step size.

Types, Objects and Operators

```
>>> "This is a sequence of 35 characters"
'This is a sequence of 35 characters’

>>> ["this", "list", "has", 4, "elements"][0]
'this’

>>> ["this", "list", "has", 4, "elements"][3]
4
```

We can get the elements from sequences with the [] operator.

So [0] returns the first element in the sequence.

[3] return the fourth element.

Types, Objects and Operators

```
>>> "This is a sequence of 35 characters"
'This is a sequence of 35 characters’

>>> "This is a sequence of 35 characters"[0]
'T'
>>> "This is a sequence of 35 characters"[3]
's'
```

We can get the elements from sequences with the [] operator.

So [0] returns the first element in the sequence.

[3] return the fourth element.

Types, Objects and Operators

```
>>> range(6)[3]
3

>>> range(6,12,2)[0]
6

>>> range(6,12,2)[2]
10

>>> range(6,12,2)[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: range object index out of range
```

range(6) is the same as [0, 1, 2, 3, 4, 5]

range(6, 12, 2) is the same as [6,8,10]

Variables

```
>>> x = 3
>>> this_is_a_longer_variable_name = 10
>>> x
3
>>> this_is_a_longer_variable_name
10
>>> x*4
12

>>> x_list = [1,2,3]
>>> x_list[2]
3
```

Recall that data is stored in memory.

We often want to remember where the data we stored is so we can use it.

To do this we use the assignment operator =.

The assignment operator gives a name to the value.

We call these variables since the value in memory they refer to can vary.

Variables

```
>>> x = 5
>>> y = 8
>>> x * y
40
>>> z = x + y
>>> z
13
>>> z = z + x
>>> z
18
```

```
>>> z = z + x
>>> z
23
>>> x = 10
>>> z
23
>>> z = z + x
>>> z
33
```

Recall that data is stored in memory.

We often want to remember where the data we stored is so we can use it.

To do this we use the assignment operator =.

The assignment operator gives a name to the value.

We call these variables since the value in memory they refer to can vary.

Modifying Variables that Name Sequences

```
>>> x = [1,2,3,4]
>>> x[1] = "a"
>>> x
[1, 'a', 3, 4]

>>> x = (1,2,3,4)
>>> x[1] = "a"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not
support item assignment
```

We can modify the contents of lists

We cannot modify the contents of tuples

Slicing Sequences

```
>>> x[1:3]
(2, 3)

>>> y = [1,2,3,4]
>>> y[2:4]
[3, 4]

>>> z="this is a string"
```

```
>>> z[3:7]
's is’

>>> z[0:3]
'thi’

>>> z[0:4]
'this’

>>> z[1:4]
'his'
```

Slicing sequences allow us to return several elements of a sequence at once.

Slicing Sequences

```
>>> x = [1,2,3]
>>> y = ["a", "b", "c"]
>>> x+y
[1, 2, 3, 'a', 'b', 'c’]

>>> "oranges" + " and " + "apples"
'oranges and apples'
```

Sequences can be concatenated with the + operator.

Order of Operations

```
>>> (2+3)+1*(4/12)
5.333333333333333

>>> ((2+3)+1)*(4/12)
2.0

>>>  4 * 2 == 8 and 4 * 2 < 8
False

>>> 7 % 2 == 1
True
```

You can use PEMDAS like in arithmetic. (Parenthesis, Exponents, Multiplication, Division, Subtraction, and then Addition)

In Python we add the requirement that operations are evaluated from left to right

P Parentheses,
then E Exponents,
then MD Multiplication and division, left to right,
then AS Addition and subtraction, left to right

Followed by Boolean operations

Converting Types

```
>>> int("2")
2
>>> float("2")
2.0
>>> int("2")
2
>>> str(2)
'2'
>>> int("this is a test")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
with base 10: 'this is a test'
```

Converting types

Sometimes we want to change the type of a variable.

int(“2”) converts a string to an integer.

Input and Output

- Everything we have done so far has been in interactive mode.
- We type one command at a time and python gives us the result immediately
- We can also run python programs in script mode by putting the source code into a text file.
- We can then have python execute all the commands in the file.
- This is how most programs are run.
- If our program needs data from the user we have to use the input and output functions (we will learn much more about functions later)

Input and Output

To read data from the user in script mode we use: input(str). We can put a string between the parentheses if we choose, and it will be printed.

To show data to the user we use the print(object) function. The object between the parentheses is displayed.

Input and Output (Examples)

```
$ python3 io.py
Enter your name:
Matthew
MatthewMatthew
```

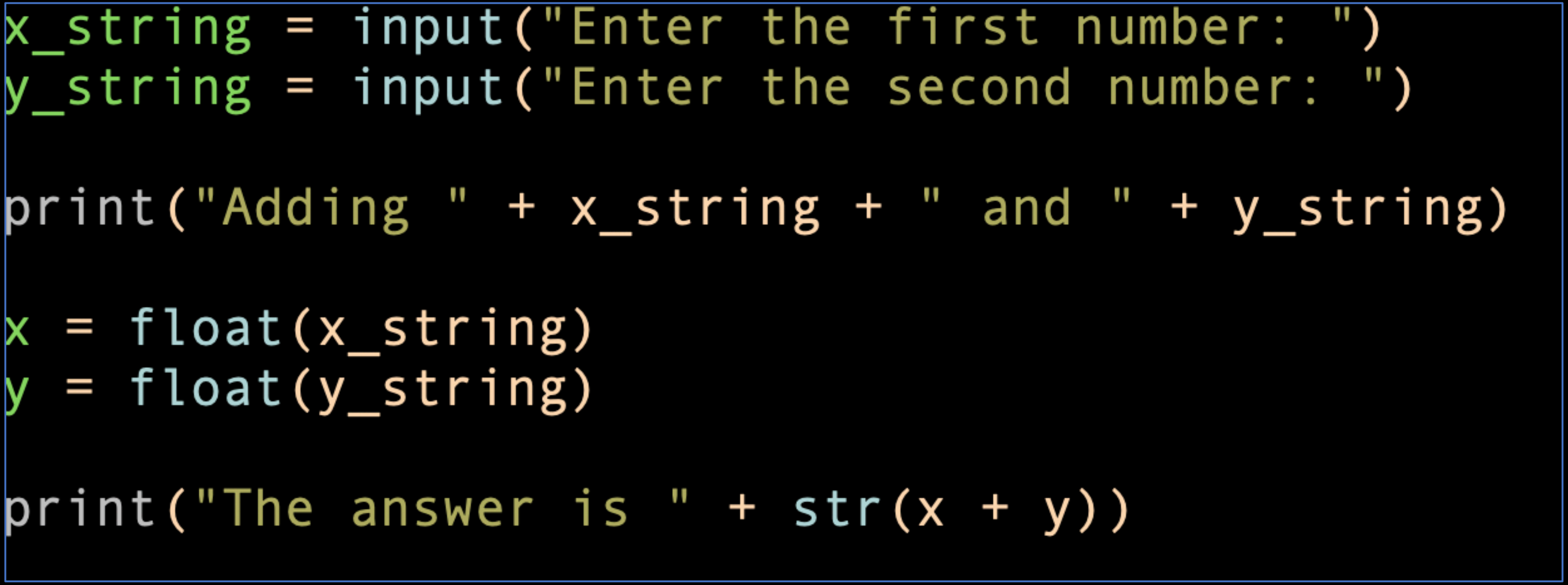
We can edit a text file with a plain text editor. Let’s save the file with the name io.py.

```
x = 2
y = input("Enter your name:”)
print(x*y)
```

Input and Output in Script Mode

```
$ python3 adder.py
Enter the first number:
5
Enter the second number:
10
Adding 5 and 10
The answer is 15.0
```

In a file named adder.py



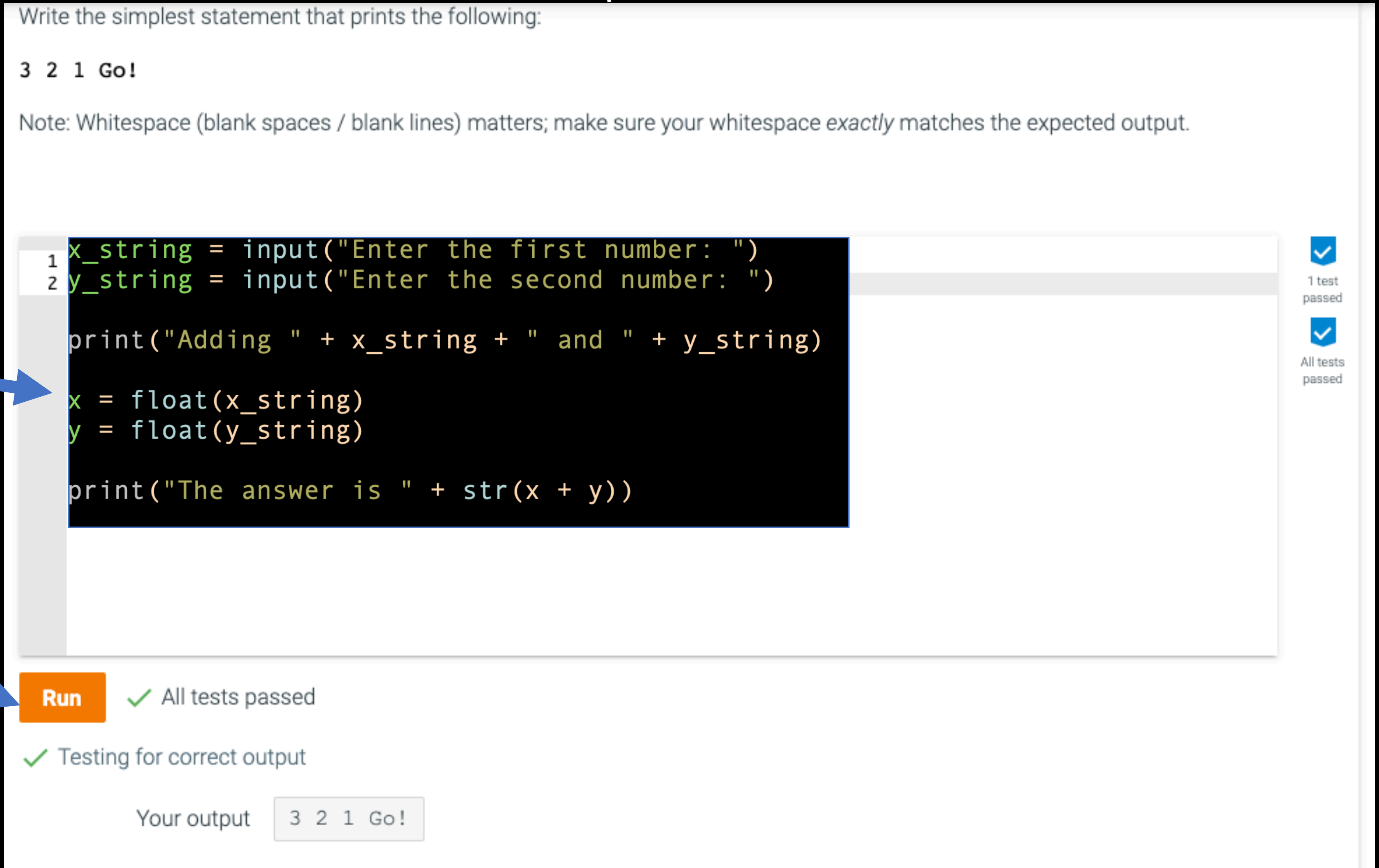
```python
x_string = input("Enter the first number: ")
y_string = input("Enter the second number: ")

print("Adding " + x_string + " and " + y_string)

x = float(x_string)
y = float(y_string)

print("The answer is " + str(x + y))
```

ZyBooks – These empty spaces act like files where you can enter python source code and run it in script mode.



Write the simplest statement that prints the following:

```
3 2 1 Go!
```

Note: Whitespace (blank spaces / blank lines) matters; make sure your whitespace *exactly* matches the expected output.

```
x_string = input("Enter the first number: ")
y_string = input("Enter the second number: ")

print("Adding " + x_string + " and " + y_string)

x = float(x_string)
y = float(y_string)

print("The answer is " + str(x + y))
```

✓ All tests passed

✓ Testing for correct output

Your output

```
3 2 1 Go!
```

adder.py

$ python3 adder.py

Enter two numbers and print True if the first is biggest and false otherwise - Line 1

Enter two numbers and print true if the first is larger - Line 2

Enter two numbers and print True if the first is larger - Line 3

Enter two numbers and print True if the first is larger - Line 4