PYTHON

LECTURE 01:
THE PYTHON INTERPRETER
THE ANATOMY OF A PYTHON SCRIPT

JIM FIX, REED COLLEGE CSCI 121

PYTHON SCRIPTING

- We start in lab tomorrow with some **Python scripting**:
  - A script is a text file containing lines of Python code.
  - Each line is a Python ***statement***.
  - The Python ***interpreter*** (the `python3` command) executes each statement, line by line, from top to bottom.
  - A statement directs that an action be made by the interpreter, which has a *state-changing* effect.

PYTHON SCRIPTING

Each Python statement directs that an action be taken that effects the system.

- Some examples of effects:
  - some text gets ***output*** (printed) to the *console*
  - some typed console ***input*** gets read and processed
  - some ***variable*** gets assigned a newly computed value
  - a window is displayed
  - a file is read
  - some web content is fetched
  - a noise is made
  - etc., etc.

RUNNING A SCRIPT

- The script text below was saved as **hello_calc.py**

```
print("Hello there, everyone!")
print("This is our first Python program.")
print("Did you know that 78687 times 89798 is this?")
print(78687 * 89798)
```

- On my Mac, within Terminal, after the **prompt**, I enter the **command**:

```
C02MX1KLFH04:examples jimfix$ python3 hello_calc.py
Hello there, everyone!
This is our first Python program.
Did you know that 78687 times 89798 is this?
7065935226
C02MX1KLFH04:examples jimfix$
```

- The Python interpreter outputs those **four lines of text**.

ANOTHER EXAMPLE: VARIABLES

- Here is that same program, slightly modified:

```
print("Hello there, everyone!")
print("This is our second Python program.")
result = 78687 * 89798
print("Did you know that 78687 times 89798 is this?")
print(result)
```

- The third line is an **assignment statement**.
  - It introduces a variable named `result` and gives it a value.
  - That value is saved in Python's memory.
  - It can be accessed by name later in the script.

ANOTHER EXAMPLE: VARIABLES

- Here is that same program, slightly modified:

```
print("Hello there, everyone!")
print("This is our second Python program.")
result = 78687 * 89798
print("Did you know that 78687 times 89798 is this?")
print(result)
```

- In line 5, we tell Python to **output the value** of `result`.

ANOTHER EXAMPLE: BUILT-IN FUNCTIONS

- Here is a different program, but somewhat similar:

```
print("Hello there, everyone!")
print("This is our third Python program.")
larger = max(78687,89798)
smaller = min(78687,89798)
print("Did you know", larger, "is bigger than", smaller, "is?")
```

- The third and fourth lines are also **assignment statements**.
  - Each of those lines uses a built-in Python function.
  - The first function `max` computes the smallest of the values it's fed.
  - The second function `min` computes the smallest of the values it's fed.

INTERACTIVE SCRIPTS

- This program is interacts with the program's user:

```
name = input("Could someone volunteer their name? ")
print("Hello there, “ + name + "!")
print("Thanks for volunteering like that.")
```

- Here is one such interaction within Terminal:

```
C02MX1KLFH04:examples jimfix$ python3 shoutout.py
Could someone volunteer their name? Audrey Bilger
Hello there, Audrey Bilger!
Thanks for volunteering like that.
C02MX1KLFH04:examples jimfix$
```

- The program has an assignment statement followed by 2 print statements.
- The assignment's right hand side uses a Python function **`input`**
- That function first outputs a **prompt string** to the console...
  - ➡ And then it reads a **string of input** typed into the console.

A CALCULATION EXAMPLE

- Consider this Python program **radius.py**:

```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("The radius of that circle is", radius, "units.")
```

- This has is 3 assignment statements and a print statement.
- The first defines/assigns the variable named `pi`.
- The second gets a floating point value (a "calculator number") as input, assigned to `area`. We compute that using an arithmetic formula.
- The functions `float` and `str` convert values of one type to values of another type.

SAME CALCULATION EXAMPLE, BUT USING LIBRARY FUNCTIONS

- Consider this Python program **radius_import.py**:

```
from math import pi, sqrt
area = float(input("Circle area? "))
radius = sqrt(area / pi)
print("The radius of that circle is", radius, "units.")
```

- Here we `import` some definitions from a Python package named `math`.
- `pi` is the name of a value.
- `sqrt` is the name of a function.
- There are packages for all sorts of useful Python libraries.

SOME ISSUES I'D LIKE TO ADDRESS

- different types of program data: `int` versus `float` versus `str`
- using functions, both built into Python and imported from a library
- operations you can apply to each type of value
- obtaining values entered by the program user with `input`
- displaying output carefully using `print`
- special characters (tab, end of line, quote, …)

Let's switch modes in how we use the Python interpreter…

INTERACTING WITH THE PYTHON INTERPRETER

- Python can be used to "live script":

```
C02MX1KLFH04:examples jimfix$ python3
>>> print(“hello”)
hello
>>> print(6 * 7)
42
>>> result = 6 * 7
>>> print(result)
42
>>>
```

- We can try a Python coding by interacting directly with the interpreter.
- We type in Python statements one at a time.
- Each line gets executed immediately.

THE INTERPRETER AS CALCULATOR

- Python can also be used to compute and display values:

```
CO2MX1KLFH04:examples jimfix$ python3
>>> 6 * 7
42
>>> result = 6 * 7
>>> result
42
>>> "hello" + " " + "there"
'hello there'
>>>
```

- We can enter Python values to be computed.
  - ➡ Python evaluates each expression and displays its value on the next line.
- A Python statement describes an action to be performed.
- These Python expressions instead describe a value to be calculated.
  - ➡ This evaluation is different than printing.

THE INTERPRETER AS CALCULATOR

- Python can be used to evaluate expressions:

```
CO2MX1KLFH04:examples jimfix$ python3
>>> 6 * 7
42
>>> result = 6 * 7
>>> result
42
>>> “hello” + “ “ + “there”
’hello there’
>>>
```

- Here, Python is acting differently. It calculates the value of the expression, then (quietly) converts that value into some readable text characters, then displays that text.

THE INTERPRETER AS CALCULATOR

- Python can be used to evaluate expressions:

```
CO2MX1KLFH04:examples jimfix$ python3
>>> 6 * 7
42
>>> result = 6 * 7
>>> result
42
>>> “hello” + “ “ + “there”
’hello there’
>>>
```

- It follows three steps:
  - **READs**: it looks at the expression entered after >>>
  - **EVALUATEs**: it performs that calculation, obtaining a value, including looking up variables’ values
  - **PRINTs**: it converts that value to some text and displays it.

THE INTERPRETER AS CALCULATOR

- Python can be used to evaluate expressions:

```
C02MX1KLFH04:examples jimfix$ python3
>>> 6 * 7
42
>>> result = 6 * 7
>>> result
42
>>> “hello” + “ “ + “there”
’hello there’
>>>
```

- This is the “**READ** - **EVALUATE** - **PRINT LOOP**” (or “**REPL**”).
- Having access to a **REPL** for a programming language is wonderful!
- It’s a big reason we teach programming in Python.

PYTHON PROVIDES SOME USEFUL FUNCTIONS...

```
>>> pow(2,3)
8
>>> abs(-3)
3
>>> abs(4 + 2)
6
>>> min(3,7)
3
>>> max(4, 10.5 + 8.3, 6)
18.8
>>> from math import sqrt, pow
>>> sqrt(2.0)
1.4142135623730951
>>> pow(2.0,4.5)
22.627416997969522
```

PYTHON PROVIDES ARITHMETIC

```
>>> 3 + 7
10
>>> 4 + 2 * 3
10
>>> (4 + 2) * 3
18
>>> 4 / 16
0.25
>>> 2 ** 4
16
>>> 0.1 + 0.2
0.30000000000000004
```

PYTHON PROVIDES ARITHMETIC

```
>>> 3 + 7
10
>>> 4 + 2 * 3
10
>>> (4 + 2) * 3
18
>>> 4 / 16
0.25
>>> 2 ** 4
16
>>> 0.1 + 0.2
0.30000000000000004
>>> type(4)
<class 'int'>
>>> type(0.25)
<class 'float'>
```

INTEGERS VERSUS FLOATING POINT NUMBERS

- Python has two types of number values: `int` and `float`
- With integers, computation is exact.
- With floating point numbers ("*floats*"), computation is approximate.

```
>>> 10 / 2
5.0
>>> 3 + 4.0
7.0
>>> int(8.7)
8
```

INTEGER VERSUS FLOATING POINT DIVISION

- With the normal division operation, the slash `/`, you get a float.

```
>>> 10.2 / 2.0
5.1
>>> 10 / 2
5.0
>>> 87 / 10
8.7
```

INTEGER VERSUS FLOATING POINT DIVISION

- There is also an integer division operation, the double slash operator `//`.
  - ➡ This gives the integer quotient.
  - ➡ The remainder due to the division is discarded.

```
>>> 10 // 2
5
>>> 87 // 10
8
```

PYTHON HAS // AND % OPERATORS

- The `//` operation ("div") gives the integer quotient due to the division of two integers:

```
>>> 345 // 12
28
```

- The `%` operation ("mod") gives the integer remainder due to the division of two integers:

```
>>> 345 % 12
9
```

- This property always holds:
  - ➡ *number* = *quotient* x *divisor* + *remainder*

```
>>> 28 * 12 + 9
345
```

EXAMPLE USES

```
>>> 345 % 10
?????????
>>> 345 // 10
?????????
>>> 6789 % 2
?????????
>>> 6790 % 2
?????????
>>> -26 % 2
?????????
>>> -76 % 10
?????????
>>> -26 // 2
?????????
>>> -76 // 10
?????????
```

EXAMPLE USES

```
>>> 345 % 10
5
>>> 345 // 10
34
>>> 6789 % 2
1
>>> 6790 % 2
0
>>> -26 % 2
0
>>> -76 % 10
4
>>> -26 // 2
-13
>>> -76 // 10
-8
```

TEXT STRINGS

- Python can store and compute with text:

```
>>> entry = input("Enter something: ")
Enter something: some thing
>>> entry
'some thing'
>>> type(entry)
<class 'str'>
>>> "hello"
'hello'
>>> type("hello")
<class 'str'>
>>> 'hello'
'hello'
>>> len(entry)
10
>>> len("hello")
10
```

- To describe a string of characters, you put those literal characters between double quotes.
- You can also use single quotes, and Python chooses to report strings that way,
  - ➡ These distinguish the text from a variable name.

STRING ARITHMETIC

```
>>> “hello” + “there”
‘hellothere’
>>> x = “hello”
>>> x = x + “ there”
>>> x
‘hello there’
>>> x = x + “ i must”
>>> x = x + “ be going”
>>> x
‘hello there i must be going’
```

STRING ARITHMETIC (CONT'D)

```
>>> "hello" * 3
'hellohellohello'
>>> 4 * "hello"
'hellohellohellohello'
>>> "hello" * 0
''
>>>
```

STRING ARITHMETIC (CONT'D)

```
>>> "hello" * 3
'hellohellohello'
>>> 4 * "hello"
'hellohellohellohello'
>>> "hello" * 0
''
>>> "hello" + 3
Error!
>>> "76" + 3
Error!
>>> "76" + str(3)
'763'
>>> int("76") + 3
79
>>> int("hello") + 3
Error!
```

SPECIAL CHARACTERS

- A backslash character `\` followed by a second character expresses special characters
  - ➡ a tab is `\t`, a new line is `\n`, a quote is `\'`, a backslash is `\\`

```
>>> z = input('What\'s your name?')
What's your name?John
>>> “Hello “ + z
'Hello John'
>>> print(“I\’ve “+str(19)+“ characters.\nSee?”)
I've 19 characters.
See?
>>> len(“I\’ve “+str(19)+“ characters.\nSee?”)
19
>>> print(“\thello\nthere”)
    hello
there
>>> print("/\\/\\/\\/\\"
/\/\/\/\
```

BACK TO PYTHON SCRIPTING

- Consider this Python program:

```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("The radius of that circle is "+str(radius)+" units.")
```

BACK TO PYTHON SCRIPTING

- Consider this Python program:

```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("The radius of that circle is "+str(radius)+" units.")
```

- This has is 3 assignment statements and a print statement.
- The first defines/assigns the variable named `pi`.
- The second gets a floating point value (a "calculator number") as input, assigned to `area`. We compute that using an arithmetic formula.
- The functions `float` and `str` convert values of one type to values of another type.

RECALL: PYTHON EXECUTION

- Let's take a look at this script:

```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("That circle's radius is "+str(radius)+".")
```

- What we know is that the Python interpreter runs the code, line by line, from the top line to the bottom line.
-

RECALL: PYTHON EXECUTION

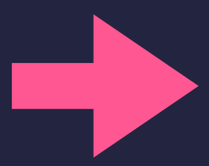
- Let's take a look at this script:

```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("That circle's radius is "+str(radius)+".")
```

- What we know is that the Python interpreter runs the code, line by line, from the top line to the bottom line.
-

RECALL: PYTHON EXECUTION

- Let's take a look at this script:

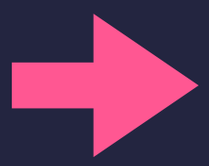


```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("That circle's radius is "+str(radius)+".")
```

- What we know is that the Python interpreter runs the code, line by line, from the top line to the bottom line.
-

RECALL: PYTHON EXECUTION

- Let's take a look at this script:



```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("That circle's radius is "+str(radius)+".")
```

- What we know is that the Python interpreter runs the code, line by line, from the top line to the bottom line.

RECALL: PYTHON EXECUTION

- Let's take a look at this script:

```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("That circle's radius is "+str(radius)+".")
```

- What we know is that the Python interpreter runs the code, line by line, from the top line to the bottom line.

RECALL: PYTHON EXECUTION

- Let's take a look at this script:

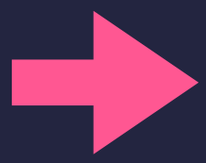
```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("That circle's radius is "+str(radius)+".")
```

- If you ever want to "watch" a Python program, try out **The Python Tutor**
  https://pythontutor.com/

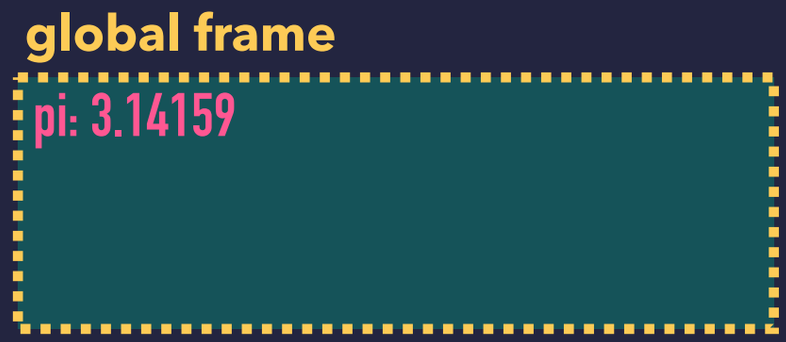
- *Using it, you'll see something like this...*

RECALL: PYTHON EXECUTION

- Let's take a look at this script:



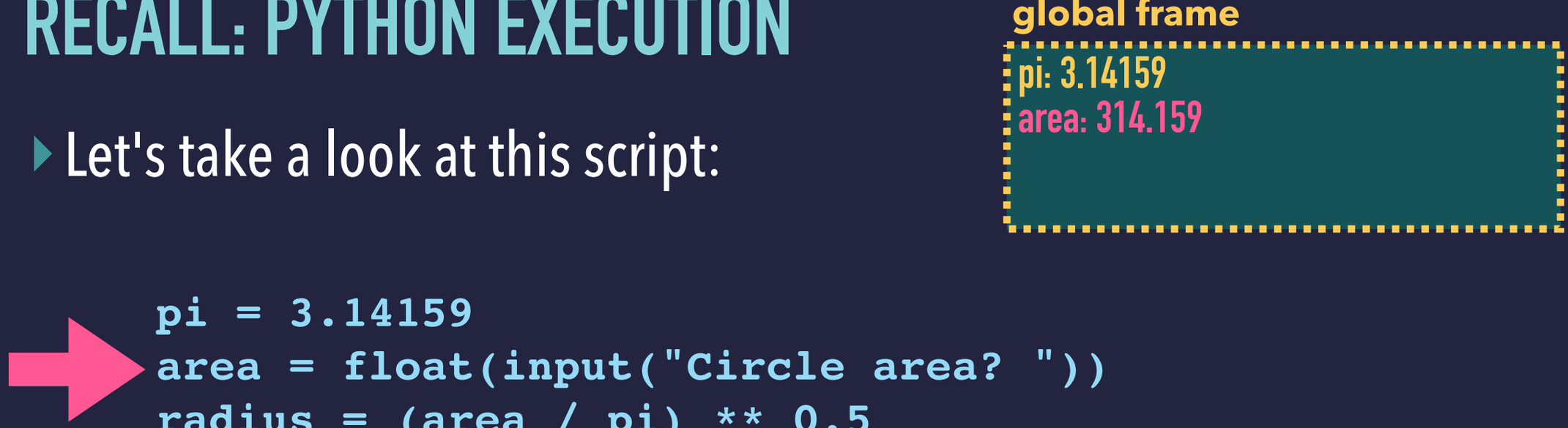
```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("That circle's radius is "+str(radius)+".")
```



- What we know is that the Python interpreter runs the code, line by line, from the top line to the bottom line.
- It also creates named memory slots for each variable that gets introduced.
  - That named slot stores a calculated value.
  - A variable's associated value is changed with each assignment statement.

RECALL: PYTHON EXECUTION

- Let's take a look at this script:

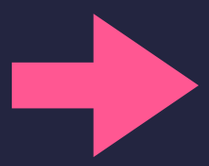



```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("That circle's radius is "+str(radius)+".")
```

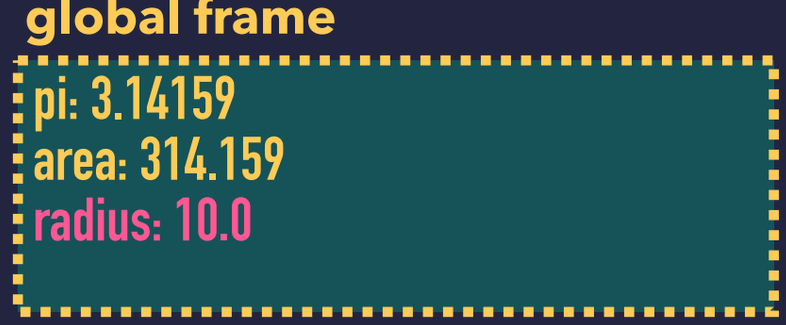
- What we know is that the Python interpreter runs the code, line by line, from the top line to the bottom line.
- It also creates named memory slots for each variable that gets introduced.
  - That named slot stores a calculated value.
  - A variable's associated value is changed with each assignment statement.

RECALL: PYTHON EXECUTION

- Let's take a look at this script:



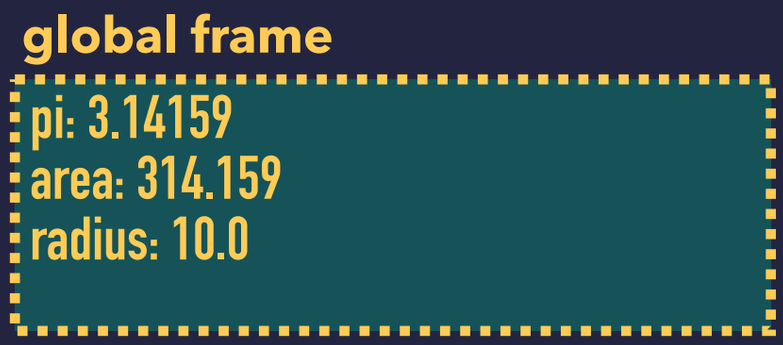
```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("That circle's radius is "+str(radius)+".")
```



- What we know is that the Python interpreter runs the code, line by line, from the top line to the bottom line.
- It also creates named memory slots for each variable that gets introduced.
  - That named slot stores a calculated value.
  - A variable's associated value is changed with each assignment statement.

RECALL: PYTHON EXECUTION

- Let's take a look at this script:

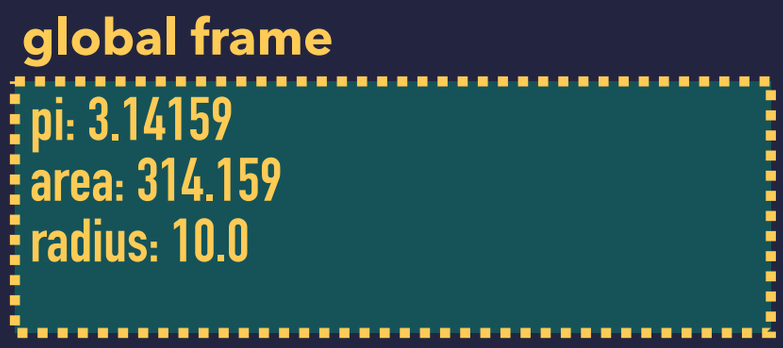



```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("That circle's radius is "+str(radius)+".")
```

- What we know is that the Python interpreter runs the code, line by line, from the top line to the bottom line.
- It also creates named memory slots for each variable that gets introduced.
  - That named slot stores a calculated value.
  - A variable's associated value is changed with each assignment statement.

RECALL: PYTHON EXECUTION

- Let's take a look at this script:




```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("That circle's radius is "+str(radius)+".")
```

- What we know is that the Python interpreter runs the code, line by line, from the top line to the bottom line.
- It also creates named memory slots for each variable that gets introduced.
  - That named slot stores a calculated value.
  - A variable's associated value is changed with each assignment statement.
    - ➡ The collection of variable slots of a script is called the ***global frame***

SUMMARY

- So far, three kinds of statements:
  - `print` statement
  - assignment statement
  - `import` statement
- Several built-in functions
  - `input`
  - conversions: `str`, `int`, `float`
  - `abs`, `min`, `max`, `pow`, and many more from the `math` library
  - `len`
  - `type`

SUMMARY (CONT'D)

- Binary operations (so far)
  - for integers: `+` `-` `*` `//` `%` `**`
  - for floats: `+` `-` `*` `/` `**`
  - for strings: `+` `*`

SUMMARY (CONT'D)

- The Python interpreter can be run *interactively* or *not*.

  - When **interactive**, you type in a statement or an expression.
    - ➡ When a statement is entered, it gets executed.
      - ✦ If there is any output, it appears on subsequent lines.
    - ➡ When an expression is entered, it gets evaluated.
      - ✦ The value that results is displayed on the next line.

  - When **not interactive**, Python just loads and runs a script.
    - ➡ Its code is executed, line by line (statement followed by statement).

TOMORROW'S LAB

- Don't forget *you have CSCI 121 lab tomorrow*!

- We'll have a lab **Homework 1** assignment:
  - assigned tomorrow, due next Tuesday 2/4, before 9am
  - the description will be at https://jimfix.github.io/csci121/assign.html
  - you'll write several Python scripts much like the examples today
  - bring your laptop

TONIGHT'S PREPARATION

- Don't forget *you have CSCI 121 lab tomorrow*!
  - ➡ And you just need to take a moment today to prepare for it…

- There is a "**Project 0**" assignment:
  - The description is linked on https://jimfix.github.io/csci121/assign.html
  - You can follow its instructions to set-up your computer for lab
  - It has three practice exercises for you to submit onto Gradescope

READINGS

- This week's lecture material can be supplemented with:
  - **Reading:** PP 1.1-1.3; TP Ch. 1 and 2; CP Ch 1.1-1.2
- In the next lecture we'll look at:
  - ➡ more scripting, focusing on formatting output and on integer division
  - ➡ the conditional statement (i.e. `if`)
  - ➡ the boolean values `True` and `False`

READINGS

- This week's lecture material can be supplemented with:
  - **Reading:** PP 1.1-1.3; TP Ch. 1 and 2; CP Ch 1.1-1.2
- In the next lecture, we'll look at:
  - ➡ more scripting, focusing on formatting output and on integer division
  - ➡ the conditional statement (i.e. `if`)
  - ➡ the boolean values `True` and `False`

"Think Python" text

"Composing Programs" text

Prof. Groce's "Principled Programming" text

TO DO

- We'll continue our exploration of Python next time.

Meanwhile, here are some things for you to do:

- Carefully read the syllabus at the course website.
- Visit the **Gradescope 121 course**.
- Complete the work of **Project 0**.
- (This sets up your computer to write/run Python code.)
- Attend lab tomorrow to start work on **Homework 1**.