

# WELCOME TO CS1!

---

- **Jim Fix, lecturer and lab instructor**

**Also: Several 121 alums as TAs**

**REED COLLEGE CSC1 121 SPRING 2025**

# COURSE OVERVIEW

---

## LECTURE 00

JIM FIX, REED COLLEGE CSC1 121

## COURSE OVERVIEW

- ▶ There is a course webpage at <http://jimfix.github.io/csci121>
  - It has the syllabus and a schedule of topics covered.
  - There I'll post lecture materials, assignments, and supplemental readings.

## ASSIGNMENT SUBMISSION THRU **GRADESCOPE**

- ▶ There is a **Gradescope** "course" for submitting completed assignments.
  - You should have received an invitation to join it.
  - You hand in homework and project work there.
- I've already posted a **Project 0** for you to work on today/tonight.
  - It will help you set up your computer.
  - It will give you practice submitting assignments.
  - Complete it before our first lab meeting tomorrow.
- Tomorrow a **Homework 1** will be started in lab, due by the next lab.
  - You'll write some basic interactive Python programs.
- ▶ I post homework descriptions at <http://jimfix.github.io/csci121/assign.html>

# CS1 IS AN INTRODUCTION TO SEVERAL THINGS

## ▶ Course topics:

- An introduction to programming. We will use the **Python** language.
- An introduction to the discipline of computer science.
- An introduction to object-oriented programming.
- An introduction to data structures and algorithms.

# CS1 IS AN INTRODUCTION TO SEVERAL THINGS

## ▶ Course topics:

- An **introduction** to programming. We will use the **Python** language.
- An **introduction** to the discipline of computer science.
- An **introduction** to object-oriented programming.
- An **introduction** to data structures and algorithms.

# CS1 IS AN INTRODUCTION TO SEVERAL THINGS

## ▶ Course topics:

- An **introduction** to programming. We will use the **Python** language.
- An **introduction** to the discipline of computer science.
- An **introduction** to object-oriented programming.
- An **introduction** to data structures and algorithms.



***No prerequisites.***

***No prior programming experience expected.***

# WHY PYTHON?

- ▶ It's a good first language.
  - It's easy to learn, loose, feature-rich.
  - Has features from several language families.
- ▶ It has a large programmer base.
  - Used by the open source community for scripting, glue.
  - Used by several scientific communities:
    - ◆ bioinformatics, computational chemistry, SAGE math.
  - Programming tools and docs are freely available.
- ▶ It's great fun.



# A PYTHON PROGRAM

```
name = input("Enter your name: ")
print("Hello, " + name + ".")

course = int(input("What's the course's #? "))
print("Ahh, yes, CSCI " + str(course) + "!")

square = 0
count = 0
while square + 2 * count + 1 <= course:
    square += 2 * count + 1
    count += 1

if course % square == 0:
    print("Did you know " + str(course))
    print("equals " + str(count) + " squared?")
```

## ANOTHER PYTHON PROGRAM

```
import time

def newton(guess, target):
    time.sleep(0.5)
    next = guess - (guess * guess - target) / (2 * guess)
    while abs(next - guess) > 0.001:
        print(guess)
        guess = next
        next = guess - (guess * guess - target) / (2 * guess)

course = 121
name = input("Enter your name: ")
print("Okay, " + name + " let me think...")
approx = newton(course/2.0, course)
print("Did you know " + str(course)
print("is roughly " + str(approx) + " squared?")
```

## WEEKLY PROGRAMMING TOPICS

- ▶ Here are the first several weeks of programming topics:
  - **WEEK 1:** scripting; program input and output; calculating things
  - **WEEK 2:** defining functions and procedures; checking conditions
  - **WEEK 3:** loops
  - **WEEK 4:** lists and dictionaries
  - **WEEK 5:** recursion
  - ...

This schedule can be found at <http://jimfix.github.io/csci121/sched.html>

# A PYTHON PROGRAM

WEEK 1: SCRIPTING

```
name = input("Enter your name: ")  
print("Hello, " + name + ".")
```



WEEK 1: INPUT

```
course = int(input("What's the course's #? "))  
print("Ahh, yes, CSCI " + str(course) + "!")
```



WEEK 1: OUTPUT

```
square = 0  
count = 0
```

```
while square + 2 * count + 1 <= course:
```

```
    square += 2 * count + 1
```

```
    count += 1
```



WEEK 1: CALCULATING

```
if course % square == 0:
```

```
    print("Did you know " + str(course))
```

```
    print("equals " + str(count) + " squared?")
```

## A PYTHON PROGRAM

```
name = input("Enter your name: ")  
print("Hello, " + name + ".")
```

```
course = int(input("What's the course's #? "))  
print("Ahh, yes, CSCI " + str(course) + "!")
```

```
square = 0
```

```
count = 0
```

```
while square + 2 * count + 1 <= course:
```

```
    square += 2 * count + 1
```

```
    count += 1
```

```
if course % square == 0:
```

```
    print("Did you know " + str(course))
```

```
    print("equals " + str(count) + " squared?")
```

*WEEK 2: LOOPS*



*WEEK 2: CHECKING CONDITIONS*



# ANOTHER PYTHON PROGRAM

```
import time
```

```
def newton(guess, target):  
    time.sleep(0.5)  
    next = guess - (guess * guess - target) / (2 * guess)  
    while abs(next - guess) > 0.001:  
        print(guess)  
        guess = next  
        next = guess - (guess * guess - target) / (2 * guess)  
  
course = 121  
name = input("Enter your name: ")  
print("Okay, " + name + " let me think ..")  
approx = newton(course/2.0, course)  
print("Did you know " + str(course)  
print("is roughly " + str(approx) + " squared?")
```

**WEEK 3: FUNCTIONS**

**WEEK 2: LOOPS**

# WEEKLY PROGRAMMING TOPICS

- ▶ Here are the first several weeks of programming topics:
  - **WEEK 1:** scripting; program input and output; calculating things
  - **WEEK 2:** checking conditions; looping
  - **WEEK 3:** defining functions and procedures
  - **WEEK 4:** lists
  - **WEEK 5:** dictionaries
  - ...

The schedule can be found at <http://jimfix.github.io/csci121/weeks.html>

- ▶ We move somewhat quickly, but it has worked well to do so!
- ▶ Though we use **Python**, these concepts are universal.
  - You are learning the structure of algorithms; ***algorithmic problem solving***.

## WEEKLY PROGRAMMING TOPICS

- ▶ The remaining weeks provide a transition to more advanced programming.
  - **WEEK 6:** recursion
  - **WEEK 7:** object-oriented programming
  - **WEEK 8:** function objects
  - ***SPRING BREAK***
  - **WEEK 9:** linked lists
  - **WEEK 10:** algorithmic efficiency; sorting and searching
  - **WEEK 11:** binary search trees
  - **WEEK 12:** file I/O and exceptions
  - **WEEK 13:** wrap-up and review



## ASSIGNMENTS

- ▶ There will be ***weekly lab homework***.
  - Assigned in Tuesday lab, due the following Tuesday before lab.
- ▶ There will be four ***programming projects***.
  - We'll give you 2-4 weeks to complete each.

## FOUR PROGRAMMING PROJECTS

- **Project 1: greed**
  - program a strategy for a two-player game of chance
- **Project 2: ciphers**
  - crack some codes
- **Project 3: hawks and doves**
  - simulate a population of birds
- **Project 4: adventure / arcade**
  - build an 80s-style game, either text-based or graphical

These will be due on occasional Thursdays.

# ASSIGNMENTS

- ▶ There will be **weekly lab homework**.
  - Assigned in Tuesday lab, due the following Tuesday before lab.
- ▶ There will be four **programming projects**.
  - We'll give you 2-4 weeks to complete each.
- ▶ There will be several **in-class quizzes**
  - Starting in a few weeks, then (roughly) every two weeks.
  - One or two short programming puzzles each.
  - Write code on paper, no use of a computer.
- ▶ There will be two **in-class exams** and a **comprehensive final**
  - Also written on paper.

# MEETING TIMES

- **LECTURE:** Mondays and Wednesdays, 80 minutes
  - 1:10-2:30pm in Library 204
- **LAB MEETING:** Tuesdays, 80 minutes
  - 10:30-11:50am in LIB 340
  - 1:40-3:00pm in LIB 340
- **EVENING TUTORING:** Sunday through Thursdays, 7-9pm in LIB 340
- **MY OFFICE HOURS:**
  - 11:20am-12:20pm Monday and Wednesday in LIB 314 (my office)
  - most Tuesday and Wednesday mornings, drop by my office!
  - most Monday and Wednesday afternoons, drop by my office!
  - other times by appointment; Thursday and Friday over Zoom

# RESOURCES

- I will post all my slides and code examples. You can probably work mostly from these.
- I'll suggest supplemental readings from **three textbooks**:
  - **Principled Programming** by Adam Groce, Reed College.
    - ◆ Closest to what I'll be teaching. Home-grown.
  - **Think Python! How To Think Like a Computer Scientist** by Allen Downey, Green Tea Press
    - ◆ Follows the topics of the course somewhat closely.
  - **COMPOSING PROGRAMS** by John DeNero, UC Berkeley
    - ◆ This is a **Python** rewrite of MIT's famous Structure & Interpretation of Computer Programs ("SICP"; uses **Scheme**)
    - ◆ Interesting supplement. Only use Chapters 1 and 2.
- All are freely available on-line.

# CS1 IS AN INTRODUCTION TO SEVERAL THINGS

- ▶ Course topics:
  - An introduction to programming. We will use the **Python** language.
  - **An introduction to the discipline of computer science.**
  - An introduction to object-oriented programming.
  - An introduction to data structures and algorithms.
  
- ▶ **Q:** What is computer science as an academic discipline?

## Q: WHAT IS COMPUTER SCIENCE?

▶ **A:** It's programming.

## Q: WHAT IS COMPUTER SCIENCE?

- ▶ **A:** It's programming.
- ▶ **A:** It's about programming.



## Q: WHAT IS COMPUTER SCIENCE?

- ▶ **A:** It's programming.
- ▶ **A:** It's about programming.
- ▶ **A:** It's about "about programming."
- ▶ Etc.
- ▶

## Q: WHAT IS COMPUTER SCIENCE?

- ▶ **A:** It's programming.
- ▶ **A:** It's about programming.
- ▶ **A:** It's about "about programming."
- ▶ Etc.
- ▶ You will learn to be reflective about programming, and also to be reflective about the tools that run programs.
- ▶ If you continue studying CS, you will learn to make tools that help people write programs. And make tools that help tools that run programs. Etc.

# PYTHON

---

## LECTURE 01:

THE PYTHON INTERPRETER

THE ANATOMY OF A PYTHON SCRIPT

JIM FIX, REED COLLEGE CSC1 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 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 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:

```
C02MX1KLFH04: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:

```
C02MX1KLFH04: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:

```
C02MX1KLFH04: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
```



## RECALL: LONG DIVISION

$$12 \overline{) 345}$$

## RECALL: LONG DIVISION

$$\begin{array}{r} 2 \\ 12 \overline{) 345} \end{array}$$

## RECALL: LONG DIVISION

$$\begin{array}{r} 2 \\ 12 \overline{) 345} \\ \underline{-24} \phantom{0} \\ 105 \end{array}$$

## RECALL: LONG DIVISION

$$\begin{array}{r} 28 \\ 12 \overline{) 345} \\ \underline{-24} \phantom{0} \\ 105 \end{array}$$

## RECALL: LONG DIVISION

$$\begin{array}{r} 28 \\ 12 \overline{) 345} \\ \underline{-24} \phantom{0} \\ 105 \\ \underline{-96} \\ 9 \end{array}$$

# RECALL: LONG DIVISION

$$\begin{array}{r} 28.9 \\ 12 \overline{) 345} \\ \underline{-24} \phantom{0} \\ 105 \\ \underline{-96} \\ 9 \end{array}$$

← the quotient

# RECALL: LONG DIVISION

A handwritten long division problem on a light blue background. The divisor is 12, and the dividend is 345. The quotient is 28, and the remainder is 9. The numbers are written in black ink. The quotient '28' is positioned above the dividend, and the remainder '9' is positioned below the final subtraction. The entire calculation is enclosed in a light blue rectangular box.

$$\begin{array}{r} 28 \\ 12 \overline{) 345} \\ \underline{-24} \phantom{0} \\ 105 \\ \underline{-96} \\ 9 \end{array}$$

← the quotient

← the remainder

## 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:

→  $number = quotient \times divisor + remainder$

```
>>> 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("/\\/\\/\\/\\/\\/\\/\\/\\/")
/\\/\\/\\/\\/
```

## AN INFORMAL QUIZ

```
>>> z = 7
>>> x = 5 + z
>>> z = z + 1
>>> print(str(z) + str(z))
??????????
>>> 0.2 + 0.1
??????????
>>> 0.2 - 0.1
??????????
>>> len('Jim\'s example:\t done.\n')
??????????
>>> print("abc\n"*4)
??????????
??????????
...?
>>> "hello" - "llo"
??????????
```





## 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 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:

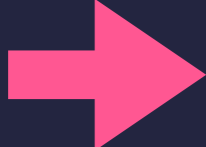
```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("That circle's radius is "+str(radius)+".")
```

- ▶ 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:



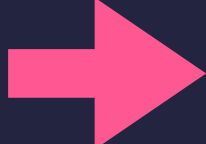
```
pi = 3.14159
area = float(input("Circle area? "))
radius = (area / pi) ** 0.5
print("That circle's radius is "+str(radius)+".")
```

- ▶ 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)+".")
```

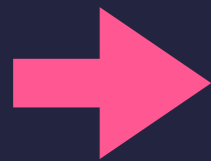
- ▶ What we know is that the Python interpreter runs the code, **line by line**, from the top line to the bottom line.



## 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:

global frame

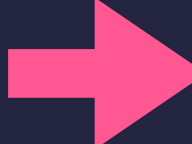
pi: 3.14159

```
➔ 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)+".")
```

global frame

```
pi: 3.14159
area: 314.159
```

- ▶ 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)+".")
```

global frame

```
pi: 3.14159
area: 314.159
radius: 10.0
```

- ▶ 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)+".")
```

global frame

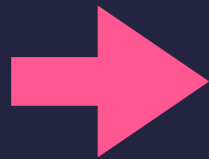
```
pi: 3.14159
area: 314.159
radius: 10.0
```

- ▶ 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)+".")
```



### global frame

```
pi: 3.14159
area: 314.159
radius: 10.0
```

- ▶ 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**

*"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.
- ▶ Join 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.
- ▶ Finish Homework 1 ***before the next Lab***.