

Lecture 1

Course Overview, Python Basics

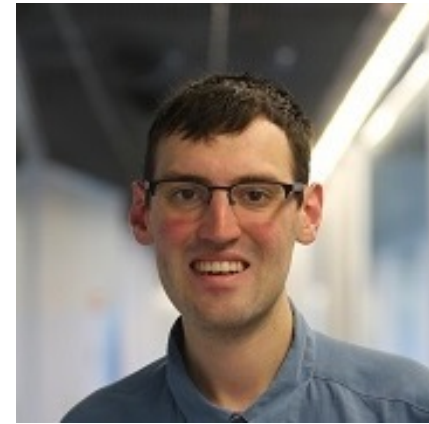
About Your Instructors

Walker White



- **Director:** GDIAC
 - **G**ame **D**esign @ **C**ornell
 - Game courses in Spring
- Teach CS 1110 in fall

Dietrich Geisler



- New instructor
 - Taught 1110 **this summer**
 - First time with a large class
- Will be back next week

We Are (Essentially) Full!

- Lecture times are somewhat flexible.
 - Technically have room for ~50 in each
 - We can use this to **swap lectures**
- Labs/sections at **fire code** capacity
 - We lost ACCEL labs last years
 - New classrooms are not as flexible
- But lots of drops; **watch Student Center!**

CS 1110 Fall 2022

- **Outcomes:**

- **Fluency** in (Python) procedural programming
 - Usage of assignments, conditionals, and loops
 - Ability read and test programs from specifications
- **Competency** in object-oriented programming
 - Ability to recognize and use objects and classes
- **Knowledge** of searching and sorting algorithms
 - Knowledge of basics of vector computation

- **Website:**

- www.cs.cornell.edu/courses/cs1110/2022fa/

Intro Programming Classes Compared

CS 1110: Python

- No prior programming experience necessary
- **No calculus**
- *Slight* focus on
 - **Software engineering**
 - **Application design**

CS 1112: Matlab

- No prior programming experience necessary
- **One semester of calculus**
- *Slight* focus on
 - **Scientific computation**
 - **Engineering applications**

But either course serves as
a pre-requisite to CS 2110

CS 1133: Short Course in Python

- 2-credit course in how to use Python
 - Material is roughly the first half of CS 1110
 - Most of the Python of 1110, but not theory
 - Two assignments; no exams
 - No experience required
- This is the only way to take Python S/U
 - CS 1110 is no longer offered S/U (in fall)
 - Best for students that just want Python

Why Programming in Python?

- Python is **easier for beginners**
 - A lot less to learn before you start “doing”
 - Designed with “rapid prototyping” in mind
- Python is **more relevant to non-CS majors**
 - NumPy and SciPy heavily used by scientists
- Python is a more **modern language**
 - Popular for web applications (e.g. Facebook apps)
 - Also applicable to mobile app development

Class Structure

- **Lectures.** Every Tuesday/Thursday
 - Not just slides; interactive demos almost every lecture
 - Because of enrollment, please stay with your section
 - **Semi-Mandatory.** 1% Participation grade from iClickers
- **Section/labs.** Phillips 318 or Hollister 401
 - Guided exercises with TAs and consultants helping out
 - Meets Tuesday/Thursday or Wednesday/Friday
 - **Only Phillips 318 has computers** (bring your laptop)
 - Contact Amy (ahf42@cornell.edu) for section conflicts
 - **Mandatory.** Missing more than 3 lowers your final grade

Class Structure

- **Lectures.** Every Tuesday/Thursday
 - Not just slides; interactive demos almost every lecture
 - Because of enrollment, please stay with your section
 - **Semi-Mandatory.** 1% Participation grade from iClickers
- **Section/labs.** Phillips 318 or Hollister 401
 - All Labs will be use the online system. But they are not intended to be “online”.
 - Contact Amy (ahf42@cornell.edu) for section conflicts
 - **Mandatory.** Missing more than 2 lowers your final grade

Is there a TextBook?

No

Is there a TextBook?

The asynchronous videos
are *essentially* the textbook

What Do I Need for this Class?

- **Laptop Computer**
 - Capable of running Python (no ChromeBooks!)
 - Minimum of 8Gb of RAM
- **Python Installation**
 - Will be using the latest Anaconda version
 - See instructions on website for how to install
- **iClicker.** Acquire by **next Tuesday**
 - Credit for answering – even if wrong
 - iClicker App for smartphone **is not** acceptable

What Do I Need for this Class?

- **Laptop Computer**

You can use computers
in Phillips 318 if needed.

- **iClicker.** Acquire by **next Tuesday**
 - Credit for answering – even if wrong
 - iClicker App for smartphone **is not** acceptable

What Do I Need for this Class?

- **Laptop Computer**
 - Capable of running Python (no ChromeBooks!)
 - Minimum of 8Gb of RAM
- **Python Installation**
 - Will be using the latest Anaconda version
 - See instructions on website for how to install
- **iClicker.** Acquire by **next Tuesday**

The only MUST purchase

This Course is OS Agnostic

Windows 10



macOS 10.15 or higher

macOS
Catalina

Do NOT Even THINK It!



Do NOT Even THINK It!



Things to Do Before Next Class

- Visit the course website:
 - www.cs.cornell.edu/courses/cs1110/2022fa/
 - This IS the course syllabus, updated regularly
- Read **Get Started**
 - Enroll in **Ed Discussions**
 - Register your **iClicker** online
 - Sign into CMS and complete **Survey 0**
 - Install Python and complete **Lab 0**
 - Take the academic integrity quiz

Things to Do Before Next Class

- Visit the course website:
 - www.cs.cornell.edu/courses/cs1110/2022fa/
 - This IS the course syllabus, updated regularly
- Read **Get Started**
 - Enroll in **Ed Discussions**
 - Register y
 - Sign into (
 - Install Pyt
 - Take the academic integrity quiz

Will talk about this
more next class

Some Words About About Grades

- This class is ***not*** curved (in traditional sense)
 - Curve = competition with other students
 - This is about material, not your classmates
- The grades mean something
 - **A**: mastered material; can be a consultant
 - **B**: good at material; ready to take 2110
 - **C**: it is a bad idea to take 2110
 - **D**: where did you go?
 - **F**: were you ever here?

Some Words About About Grades

- But this is **not** a weed-out course
 - We know students have different backgrounds
 - Students can do well regardless of experience
- But you may have to work hard!
 - If no experience, budget 10-12 hours of homework a week

	A	B	C	D/F	
All Students	40%	40%	18%	2%	
Some Experience	37%	41%	20%	2%	42%
No Experience	32%	42%	24%	2%	28%


Some Words About About Grades

- But this is **not** a weed-out course
 - We know students have different backgrounds
 - Students can do well regardless of experience
- But you may have to work hard!
 - If no experience, budget 10-12 hours of homework a week

	A	B	C	D/F	
All Students	40%	40%	18%	2%	
Some Experience	37%	41%	20%	2%	42%
No Experience	32%	42%	24%	2%	28%
Freshmen, No Exp	37%	39%	24%	0%	

Getting Started with Python

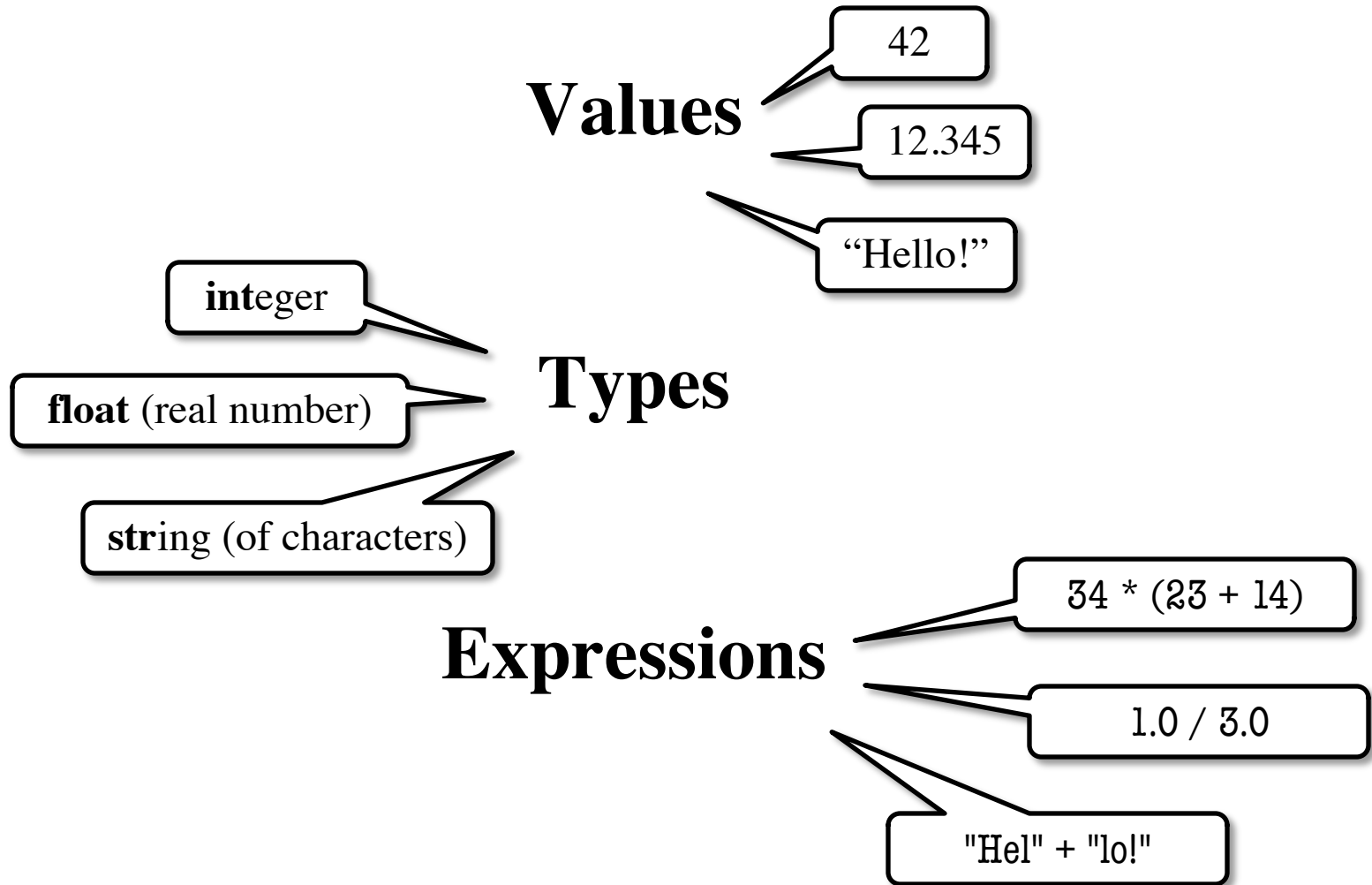
- Will use the “command line”
 - OS X/Linux: **Terminal**
 - Windows: **PowerShell**
 - Purpose of the first lab
- Once installed type “python”
 - Starts an *interactive shell*
 - Type commands at >>>
 - Responds to commands
- Use it like a calculator
 - Use to evaluate *expressions*



```
wmwhi
Last login: Sun Aug 21 14:06:34 on tt
[wmwhite@Rlyeh]:~ > python
Python 3.9.12 (main, Apr 5 2022, 01:
[Clang 12.0.0 ] :: Anaconda, Inc. on
Type "help", "copyright", "credits" o
>>> 1+1
2
>>> 'Hello'+'World'
'HelloWorld'
>>> 
```

This class uses Python 3.9

The Basics



Expressions and Values

- An **expression** represents something
 - Python *evaluates it*, turning it into a **value**
 - Similar to what a calculator does
- Examples:

>>> 2.2

Expression

(Literal)

2.2

Value

>>> (3 * 7 + 1) * 0.1

Expression

(Complex)

2.2

Value

What Are Types?

- Think about + in Python:

```
>>> 1+2
```

```
3
```

```
>>> "Hello"+"World"
```

```
"HelloWorld"
```

adds numerically

glues together

- Why does + given different answers?
 - + is different on data of different *types*
 - This idea is fundamental to programming

What Are Types?

A **type** is both

- a set of *values*, and
- the *operations* on them

Example: **int**

- **Values:** integers
 - ..., -1, 0, 1, ...
 - Literals are just digits:
1, 45, 43028030
 - No commas or periods
- **Operations:** math!
 - +, - (add, subtract)
 - *, // (mult, divide)
 - ** (power-of)

Example: **int**

- **Values:** integers
 - ..., -1, 0, 1, ...
 - Literals are just digits:
1, 45, 43028030
 - No commas or periods
- **Operations:** math!
 - +, - (add, subtract)
 - *, // (mult, divide)
 - ** (power-of)
- **Important Rule:**
 - **int** ops make **ints**
 - (if making numbers)
- What about division?
 - 1 // 2 rounds to 0
 - / is **not** an **int** op
- Companion op: %
 - Gives the remainder
 - 7 % 3 evaluates to 1

Example: float

- **Values:** real numbers
 - 2.51, -0.56, 3.14159
 - Must have decimal
 - 2 is **int**, 2.0 is **float**
- **Operations:** math!
 - +, - (add, subtract)
 - *, / (mult, divide)
 - ** (power-of)
- Ops similar to **int**
- **Division** is different
 - Notice /, not //
 - 1.0/2.0 evals to 0.5
- But includes //, %
 - 5.4//2.2 evals to 2.0
 - 5.4 % 2.2 evals to 1.0
- Superset of **int**?

float values Have Finite Precision

- Try this example:

```
>>> 0.1+0.2  
0.30000000000000004
```
- The problem is **representation error**
 - Not all fractions can be **represented** as (finite) decimals
 - **Example**: calculators represent $2/3$ as 0.666667
- Python does not use decimals
 - It uses IEEE 754 standard (beyond scope of course)
 - Not all decimals can be **represented** in this standard
 - So Python picks something close enough

float values Have Finite Precision

- Try this example:

```
>>> 0.1+0.2
```

```
0.30000000000000004
```

- The problem is that **Expressions vs Values** decimals
 - Not all floats are exact decimals
 - **Example**

- Python does not use decimals
 - It uses IEEE 754 standard (beyond scope of course)
 - Not all decimals can be **represented** in this standard
 - So Python picks something close enough

int versus float

- This is why Python has two number types
 - **int** is **limited**, but the answers are always **exact**
 - **float** is **flexible**, but answers are **approximate**
- Errors in float expressions can propagate
 - Each operation adds more and more error
 - Small enough not to matter day-to-day
 - But important in scientific or graphics apps (high precision is necessary)
 - Must think in terms of **significant digits**

Using Big float Numbers

- **Exponent notation** is useful for large (or small) values

- $-22.51e6$ is $-22.51 * 10^6$ or -22510000
- $22.51e-6$ is $22.51 * 10^{-6}$ or 0.00002251

A second kind
of **float** literal

- Python *prefers* this in some cases

```
>>> 0.000000000001  
1e-11
```

Remember: values
look like **literals**

Example: **bool**

- **Values:** True, False
 - That is it.
 - Must be capitalized!
- **Three Operations**
 - **b and c**
(True if **both** True)
 - **b or c**
(True if **at least one** is)
 - **not b**
(True if b is **not**)
- Made by **comparisons**
 - **int**, **float** operations
 - But produce a **bool**
- Order comparisons:
 - $i < j$, $i \leq j$
 - $i \geq j$, $i > j$
- Equality, inequality:
 - $i == j$ (**not** =)
 - $i != j$

Example: **str**

- **Values:** text, or *sequence of characters*
 - String literals must be in quotes
 - Double quotes: "Hello World!", " abcx3\$g<&"
 - Single quotes: 'Hello World!', ' abcx3\$g<&'
- **Operation:** + (catenation, or concatenation)
 - 'ab' + 'cd' evaluates to 'abcd'
 - concatenation can only apply to strings
 - 'ab' + 2 produces an **error**