CS134:
Python Types and Expressions
Announcements & Logistics

- **HW 1** due today at 11 pm (Google form)
- **Lab 1** today/tomorrow on Zoom, due Wed/Thur at 10pm
  - Mon/Tue 1:10 pm: Rohit/Jeannie & Steve
  - Mon/Tue 2:35 pm: Steve
- Goal: Setup computers, gain experience with the workflow and tools
- Start with some short and sweet Python programs

- **Office hours and TA hours start today**
  - Check calendar on webpage for hours

- **Questions??**
Last Time

- Discussed course logistics
- Important take-aways:
  - **Setup** your personal machine soon (setup guides on course webpage)
    - If you get stuck, we’ll help you in lab!
  - **Review** syllabus and check out course webpage
Today’s Plan

- Discuss **data types** and **variables** in Python
  - `int`, `float`, `boolean`, `string`
- Learn about basic **operators**
  - arithmetic, assignment
- Experiment with built-in Python **functions** and expressions
  - `int()`, `input()`, `print()`
- Investigate different ways to run and interact with Python
Aspects of Languages

• **Primitive constructs**
  • English:
    • words, punctuation
  • Programming languages:
    • numbers, strings, simple operators
Aspects of Languages

• **Syntax**
  - **English:**
    - “boy dog cat” (incorrect), “boy hugs cat” (correct)
    - “Let’s eat grandma!” (probably incorrect), “Let’s eat, grandma!” (correct)
  - **Programming language:**
    - “hi”5 (incorrect), 4*5 (correct)
Aspects of Languages

- **Semantics** is the meaning associated with a syntactically correct string of symbols.

- **English:**
  - Can have many meanings (ambiguous), e.g.
  - “Flying planes can be dangerous”
  - Other examples?

- **Programming languages:**
  - Must be unambiguous
  - Can only have one meaning
  - Actual behavior is not always the intended behavior!
Python3

• Programming language used in this course
• Great introductory language
  • Better human readability and user friendly syntax
• For this class, we need Python 3.9.1 or above
• Checking version of Python on machine
  • (Mac, Linux, or Windows Subsystem for Linux)
  • Type `python3 --version` in Terminal (Ubuntu Shell)
• **Preinstalled on all lab machines**
• Installing Python3 on your machine: see setup guide on webpage
Python Primitive Types

• Every **value** has a data **type**. For example:
  • 10 is an integer (type: **int**)
  • 3.145 is a decimal number (type: **float**)
  • ‘Williams’ or “Williams” is a sequence of characters (type: **string**)
  • 0 (False) and 1 (True) (type: **boolean or bool**)
    • Represent answers to decision questions (yes/no)
  • “Empty” value (type: **None**)
• We will revisit booleans and None types soon!

Knowing the **type** of a **value** allows us to choose the right **operator** for expressions.
Python Operators

• **Arithmetic operators:**
  • + (addition), - (subtraction), * (multiplication)
  • / (floating point division, returns a value with a decimal point)
  • // (integer division, returns an integer)
  • % (modulo, or remainder)
  • ** (power, or exponent)

(We will try these out with examples later and see how they behave)

• **Assignment operator:**
  • = ("is assigned", not "equals")
  • Not to be confused with mathematical equality, which is written as == in programming languages
  • = is used to "assign" values to **variables**
Variables and Assignments

- A **variable** names a value that we want to use later in a program
  - If we define `num = 17` then the value 17 essentially gets stored in a box in memory with the label `num`
  - We are **assigning** `num` (a variable) the value 17
- Once defined, we can reuse variable names again, and later assignments can change the value in a variable box
  - `num = num - 5`
  - What is stored in `num` after this evaluates?

**Math vs Programming.** An assignment: expression on the right evaluated first and the value is stored in the variable name on the left
Variables and Assignments

• A **variable** names a value that we want to use later in a program
  • If we define `num = 17` then the value **17** essentially gets stored in a box in memory with the label `num`
  • We are **assigning** `num` (a variable) the value **17**
• Once defined, we can reuse variable names again, and later assignments can change the value in a variable box
  • `num = num - 5`
  • What is stored in `num` after this evaluates?
  • `var = <expression>` (result of expression gets stored in the variable box `var`)
• **Question.** Why would we want to name values or expressions?
Abstracting Expressions

• Why give names to data values or the results of expressions?
  • To reuse names instead of values
  • Easier to change code later
• For example:

```python
pi = 3.1415926  # useful to name
radius = 2.2
area = pi * (radius**2)
# suppose now we want to change radius
radius = 2.2 + 1
area = pi * (radius**2)  # new area
```
An Aside: Python Interfaces

• Now we know a little bit about
  • Python primitive data types (ints, floats, strings, etc)
  • Operators (mathematical, assignment)
  • Variables
• Before we move on to more concepts, let's experiment a bit to see what we can do with these
• This semester, we will run Python code in two ways:
  • As a **script** (save code in a file, run from Terminal)
  • **Interactively** (from Terminal) in an interactive python session
Python: Program as a Script

- A **program** is a sequence of definitions and commands
- Definitions are evaluated
- Commands are executed and instruct the interpreter to do something
- Type instructions in a **file** that is read and evaluated sequentially
  - For example, this week in lab you will write `helloworld.py` in a file and then execute it from the Terminal with `python3 helloworld.py`
  - Common method: good for longer pieces of code or programs
  - We will use this method in our labs
  - Called "running the Python program as a script"
Python: Interactive

• Running Python **interactively** is great for introductory programming
• Launch the Python interpreter by typing `python3` in the Terminal
  • Opens up Interactive Python
  • Almost like a "calculator" for Python commands
  • Takes a Python expression as input and spits out the results of the expression as output
  • Great for trying out short pieces of code
  • Great for teaching Python in Lectures
• Today we will use a "fancy" version of Interactive Python called **Jupyter Notebooks**
Types and Expressions

Jupyter Notebooks provide a rich interface to interactive Python. To read more about how to use them, check out our [How To Jupyter](#) guide.

Types in Python

The built-in `type()` function lets us see the data type of various values in Python.

Note: The one line phrases after # are comments, they are ignored during execution.

```
In [1]: type(134)
In [2]: type('134')  # single quotes
In [3]: type("134")  # double quotes
In [4]: type(3.14159)
In [5]: type('')
In [6]: type(0)
In [7]: type(False)
```
Python Built-In Functions
Built-In Functions

• Python comes with a ton of built-in capabilities in the form of functions
• We’ll formally discuss functions soon, but for now, let’s look at a few examples
Built-in functions: input()

- `input()` displays its single argument as a prompt on the screen and waits for the user to input text, followed by `Enter/Return`
- It returns the entered value as a `string`

```python
In[1] input('Enter your name: ')  
Enter your name: Harry Potter  
Out[1] 'Harry Potter'
In[2] age = input('Enter your age: ')  
Enter your age: 17  
In[3] age  
Out[3] '17'
```

Prompts in Maroon. User input in blue. Inputted values are by default a `string`
Built-in functions: print()

- `print()` displays a character-based representation of its argument(s) on the screen and returns a special `None` value (not displayed).

```
In[1] name = 'Harry Potter'
In[2] print('Your name is', name)
Your name is Harry Potter
In[3] age = input('Enter your age : ')
Enter your age: 17
In[4] print('The age of ' + name + ' is ' + age)
The age of Harry Potter is 17
```

Comma as a separator adds a space

Can also add spaces through string concatenation
Built-in functions: int()

- When given a string that's a sequence of digits, optionally preceded by +/-, int() returns the corresponding integer
- On any other string it raises a ValueError
- When given a float, int() returns the integer that results after truncating it towards zero
- When given an integer, int() returns that same integer

In [1] int('42')
Out [1] 42
In [2] int('-5')
In [3] int('3.141')
ValueError
Built-in functions: float()

- When given a string that's a sequence of digits, optionally preceded by +/-, and optionally including one decimal point, float() returns the corresponding floating point number.
- On any other string it raises a ValueError
- When given an integer, float() converts it to a floating point number.
- When given a floating point number, float returns that number

```python
In[1] float('3.141')
Out[1] 3.141
In[2] float('−273.15')
Out[2] −273.15
In[3] float('3.1.4')
ValueError
```
Built-in functions: `str()`

- Converts a given type to a **string** and returns it.
- Returns a syntax error when given invalid input.

```
In[1] str(3.141)
Out[1] '3.141'
In[2] str(None)
Out[2] 'None'
In[3] str(134)
Out[3] '134'
In[4] str($)
SyntaxError: invalid syntax
```
An Aside: Submitting Labs via Git

- Git is a version control system that lets you manage and keep track of your source code history

- GitHub is a cloud-based git repository management & hosting service

  - **Collaboration**: Lets you share your code with others, giving them power to make revisions or edits

- GitLabs is similar to GitHub but we maintain it internally at Williams and will use to handle submissions and grading
An Aside: Directories in Unix

- 'Folders' on your computers are called 'directories' in Unix-based operating systems

- Your ‘current directory’ is important when executing commands on the Terminal

- For example, programs that run as a script, such as `helloworld.py`, must be in the same directory as where you execute the command `python3 helloworld.py`

- Otherwise your computer doesn’t know which program to run

- Similarly, when you `git pull`, you need to be in the correct directory

- Useful to learn how to navigate between directories with the Terminal
Useful Unix Commands

- **pwd** print working directory
- **mkdir <dir name>** make new directory (or folder)
- **cd <dir name>** change directory
- Special directory names
  - . (single dot, current directory)
  - .. (two dots, parent directory)
  - ~ (tilde, home directory)
- **cd ..** takes you to the parent directory
- **cd** takes you “home”
- **ls** shows contents of current directory