CS134:
Python Types and Expressions

Announcements/ Logistics

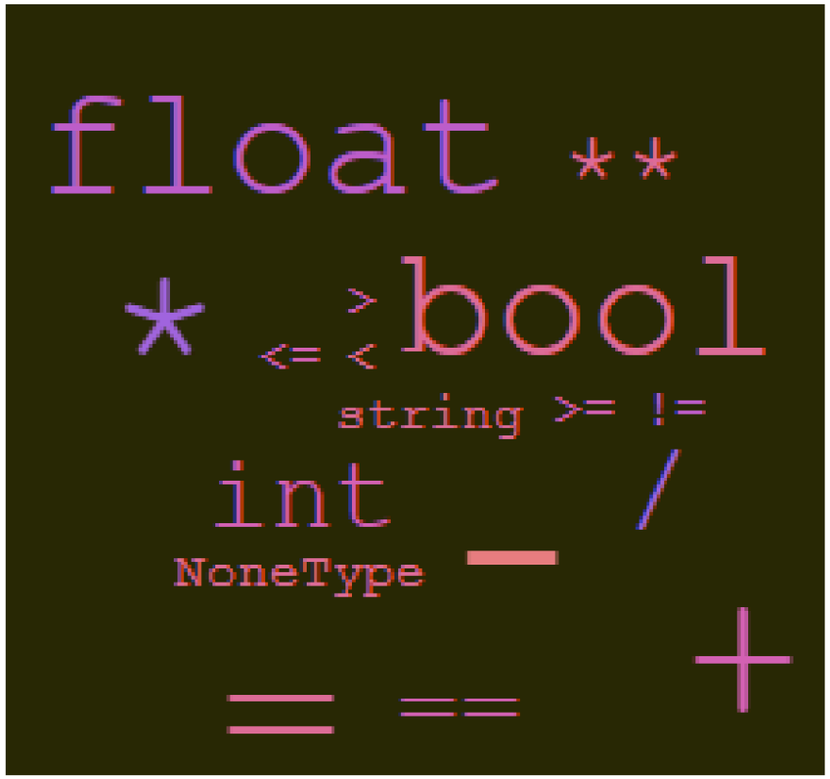
- **Homework 1** due today at 10 pm (Google form)
- Scheduled labs this week:
  - Monday1.10 pm: TCL 217A - Shikha), TCL 216 - Kelly
  - Monday 2.35 pm: TCL 217A - Kelly
  - Tuesday 1.10 pm: TCL 217A - Jeannie), TCL 216 - Kelly
  - Tuesday 2.35 pm: TCL 217A - Kelly
- **Office hours (today):**
  - Shikha 3-5 pm, TCL 204
- TA hours (today)
  - 7-11 pm in TCL 217A and TCL 216
- Goal for this week: meet at least two TAs & talk to at least one instructor outside class!

This Week

- **Homework 1** due today at 10 pm (Google form)
- **Setup** your personal machine:
  - Find the Mac and Windows Setup Guide on course page
  - Try out all the steps
  - If you get stuck, come to us!
  - Please do this soon! First week is the best time to get this done
- Read about CS 134 Tools (also linked under Resources)
- **Lab 1** (start during lab session)
  - Goal: get you comfortable with the workflow and tools
  - Start with some short and sweet Python programs
  - Get used to different interfaces we will see in this course

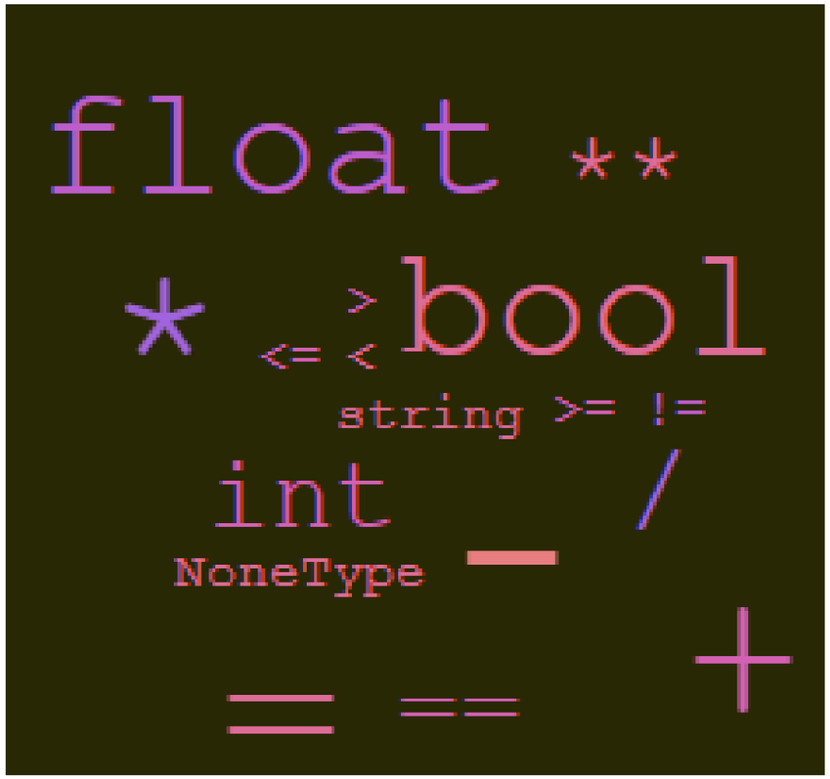
Aspects of Languages

- **Primitive constructs**
  - English: words
  - Programming languages: numbers, strings, simple operators



Aspects of Languages

- **Syntax**
  - English: "cat dog boy" (incorrect), "cat hugs boy" (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 can sometimes be not what is intended !

Python3

- Programming language in this course
- Great introductory language
  - Better human readability and user friendly
- For this class, we need `Python 3.6.4` or above
- Checking version of Python on machine
  - (Mac, Linux, or Windows Subsystem for Linux)
  - Typing python --version in Terminal (Ubuntu Shell)
- **Preinstalled on all lab machines**
- Installing Python3 on your machine: setup guide

Python: Program as a Script

- A **program** is a sequence of definitions and commands
  - Definitions are evaluated
  - Commands are executed and instruct interpreter to do something
- Can be typed in a **file** that is read and evaluated at the terminal
  - For example, we write helloworld.py in a file and then executed it from the Terminal with python3 helloworld.py
  - Standard method: good for longer pieces of code
  - We will use this method in labs as well
  - Called "running the Python program as a *script*"

Python: Interactive

- What makes Python great for introductory programming:
  - Interactive language
- Can 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 an 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

Let us Look at

Lecture 2: Jupyter Notebook

- **Need to add somewhere:**
- In programing, a sequence of commands is read left to right, and top down in sequence

Python Commands

- **Commands** instruct the Python interpreter to do something
- Can be typed directly into **Interactive Python** or stored in a file that is read and evaluated
- Let us look at some

Python Primitive Types

- Each **value** has a type, for example
  - E.g. 10 is an integer (type: int)
  - 3.145 is a decimal number (type: float)
  - 'Williams' is a sequence of letters (type: string)
  - Special type in programming: 0 and 1 (type: bool)
  - Special type in programming: None (NoneType)
- Can use command `type()` to ask Python to tell us the type of a value

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

Python Primitive Types

- Each **value** has a type, for example
  - E.g. 10 is an integer (type: int)
  - 3.145 is a decimal number (type: float)
  - 'Williams' is a sequence of letters (type: string)
  - Special type in programming: 0 and 1 (type: bool)
  - Special type in programming: None (NoneType)
  - E.g. `int, float, str, bool, NoneType`
  - Can use `type()` to see the type of an value
  - Knowing the **type** of a value allows us to choose the right **operator** when creating **expressions**
- **Operators**:
  - E.g. `+ - * / % // =`
- **Expressions**:
  - E.g. `'3+4'`, `'Williams' * 3`, `len('shikha')`
  - Always produce a value as a result
- **Built-in functions**:
  - `int, float, str, print, input, max, min, len`

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

Python Primitives

- **Values**:
  - E.g. 10 (integer), 3.145 (float), 'Williams' (string)
- **Types**:
  - E.g. `int, float, str, bool, NoneType`
  - Can use `type()` to see the type of an value
  - Knowing the **type** of a value allows us to choose the right **operator** when creating **expressions**
- **Operators**:
  - E.g. `+ - * / % // =`
- **Expressions**:
  - E.g. `'3+4'`, `'Williams' * 3`, `len('shikha')`
  - Always produce a value as a result
- **Built-in functions**:
  - `int, float, str, print, input, max, min, len`

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

Python Program

- A **program** is a sequence of definitions and commands
  - Definitions are evaluated
  - Commands are executed by the Python interpreter in a shell
- **Commands** instruct interpreter to do something
- Can be typed directly in a shell or stored in a **file** that is read and evaluated
  - In lectures, we'll use Jupyter for instant evaluation and output
  - In labs, you'll write your program as a script and save it with a .py extension, e.g. `` `hellowold.py'. `` You can execute the program from the terminal: `python3 helloworld.py`

Python and Interfaces

- Interfaces we will use to Python:
  - **IPython**
    - Interactive command-line terminal for Python
    - Created by Fernando Perez
    - Powerful interface to use Python
    - Often called a **REPL ('Read-Eval-Print-Loop')**
  - **Jupyter Notebook**
    - Created in 2011, a new web-based interface for Python
    - Teaching aid in class—makes teaching programming more interactive and efficient
    - Also Popular tool for scientific exposition, especially data science (even in languages such as R and Julia)
- In labs you will be writing python programs as a script with extension .py that can be executed from the terminal

Python: Interactive Ways

">>" tells you it is an interactive python session in the terminal

```
>> 1 + 2
3
>> 3* 4
12
```

"In [] and Out" tells you it is an interactive python session in Jupiter

```
In [10]: 12/3
Out [10]: 4.0
```

Out vs Print: "Print" means it is printed onto the console and will actually be shown to the user when you edit/run the script

```
In [11]: print(25//5)
5
```

Operator Precedence

- Operator precedence without parenthesis

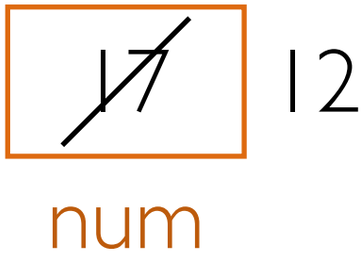
  `**`
  `*`
  `/`
  `+` and `-` (left to right as they appear)

- Parenthesis used to override precedence and tell Python do these operations within parenthesis first

Variable Assignment

- A variable names a value that we want to use later in a program
- **Variables as a box model.**
  An assignment statement `var = exp` stores the value of `exp` in a "**box**" labeled by the variable name
- Later assignments can change the value in a variable box. **Note:** The symbol '=' is pronounced **"gets" not "equals"**!

```
In [1] num = 17
In [2] num
Out [2] 17
In [3] num = num - 5
In [4] num
Out [4] 12
```



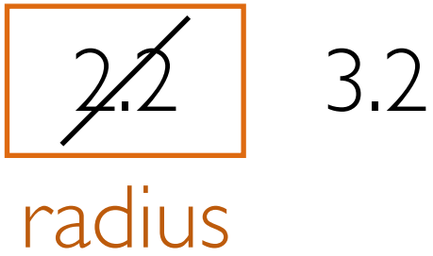
Abstracting Expressions

- Why give names to values of expressions?
- To reuse names instead of values
- Easier to change code later

```
In [1] pi = 3.14159
In [2] radius = 2.2
In [3] area = pi * (radius**2)
In [4] area
Out [4] 15.205295600000001
In [5] round(area, 2)
Out [5] 15.21
```

Programming vs Math

- In programming, "we don't solve for x"



3.2

```
pi = 3.14159
radius = 2.2
area = pi * (radius**2)
radius = radius + 1   #can be shortened to radius +=1
```

An assignment: expression on the right evaluated first and the value is stored in the variable name on the left

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**.

```
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
```

Printed on the console; 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` (correct type, but wrong value of that type).
- When given a float, `int` return the integer the results by truncating it toward zero.
- When given an integer, `int` returns that integer.

```
In [1] int('42')
Out [1] 42
In [2] int('-5')
Out [2] -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 floating point number.
- When given a floating point number, float returns that number.

```
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
```

Expressions vs Statement

Expressions

- They always produce a value

```
10 + 12 - 3
num + 4
“CS” + “134”
```

- Expressions can be composed of any combination of values, variables, and function calls

```
max(10, 20)
```

Statements

- They perform an action (that can be visible, invisible or both)

```
age = 12
print(‘Hello World’)
```

- Statements may contain expressions, which are evaluated **before** the action is performed

```
print('She is ' + str(age) + '
years old')
```

- Some statements return a **None** value which is not normally displayed

Error Messages

- **Type Errors**

  ‘134’ + 5

  len(134)

- **Value Errors**

  int(‘3.142’)

  float(‘pi’)

- **Name Errors**

  int(‘3.142’)

  float(‘pi’)

- **Syntax Errors**

  2ndValue = 25

  1 + (ans = 42)

Submitting Labs: 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