

Programming Basics

Announcement

- Our peer tutor Yantan (Ian) Mei can be reached via yantao.mei@dukekunshan.edu.cn
- Trailokaya (Raj) Bajgain can be reached via trailokaya.bajgain@dukekunshan.edu.cn
- 4pm on Friday as a Drop-in session?

Learning Objectives

- Recognize and use the basic data types in programs
- Interpret and react to basic error messages caused by programs
- Use variables in code and trace the different values they hold

Python and Jupyter Notebook

Programs are Algorithms for Computers

Computers only know how to do what we tell them to do. Programs communicate with a computer and tell it what to do.

Algorithms can be expressed as programs in many different programming languages. Different languages use different syntax (wording) and commands, but they all share the same set of algorithmic concepts.

In this class, we'll use Python, a popular programming language.

Python is Simple and Highly Useful



The Python programming language is designed to be easy to read and simple to implement algorithms in.

There are also a huge number of libraries that implement useful things in Python. We'll use libraries that support graphics, data analysis, randomness, and more.

Python's main weakness is efficiency – it can be slower than other languages. But that won't matter for our purposes.



Jupyter Notebook and Colab

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.

The Jupyter Notebook combines two components:

A web application: a browser-based tool for interactive authoring of documents.

Notebook documents: a representation of all content visible in the web application.

Developing Environment for Python

When writing programs, we use IDEs – Integrated Development Environments. These are like text editors for programs.

In this class, we recommend that you use Google Colab, a free, web-based Jupyter Notebook. It does not require any software installation, which makes it good for novices.

However, you can also use IDEs that you prefer, such as PyCharm, Pyzo, or even text editors.

Data Types

Data Is Information We Can Manipulate

- Most programs we write will keep track of some kind of information and change it with actions. We call that information data.
- Data have different types depending on their properties. We'll start by going over three core types: numbers, text, and truth values.

Numbers and Operations in Python

Integers (0, 14, -7) are whole numbers.

Floating point numbers (3.0, 5.735, 8e10) include a decimal point.

`+` : addition

`-` : subtraction

`*` : multiplication

`/` : division

`**`: power ($2^3 = 8$)

`()` : use parentheses to specify the order to evaluate operations

An expression like `4**2` or `(5-2)/3` is a piece of code that evaluates to a data value.

Text in Python

Text values in Python are called strings, for reasons we'll go over later. Text is recognized by Python when it is put inside of quotes, either single quotes ('Hello') or double quotes ("Hello").

Strings can be concatenated together using addition. E.g,

"Hello" + "World" produces "HelloWorld".

Truth Values in Python

Finally, Python can evaluate whether certain expressions are true or false. These types of values are called Booleans after the mathematician George Boole.

Booleans can be either `True` or `False` (no quotes, and capitals are required). These names are built into Python directly.

To get a Boolean, we can write `True` or `False` directly, or do a comparison. The basic comparison operators are familiar: `<`, `>`, `<=`, and `>=`.

We can also check if two values are equal (`==`), or not equal (`!=`).

Eg., `"Hello" == "World"` evaluates to `False`

Type Mismatches Cause Errors

Be careful when mixing types in Python, as that can cause error messages. An error message is how the computer tells you it doesn't understand a command you wrote.

For example, `"Hello" + 5` results in a `TypeError`.

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: must be str, not int
```

Similarly, `"Hello" < True` results in a `TypeError`.

On the other hand, integers and floating point numbers can be mixed. When this happens, the result is usually a floating point number.

For example, `8 * 2.0` results in `16.0`

Data Type Names

When reading error messages, note that Python uses shortened names for the four types we've covered.

Integers are called `int`

Floating point numbers are called `float`

Strings are called `str`

Booleans are called `bool`

Writing Code in Files

Writing Longer Programs: Use the Editor

What if we want to run more than one line of code at a time? We'll need to use the editor.

IDEs will interpret the entire text file into Python code the computer will understand. It will then run line-by-line through the entire program sequentially. This is different from the interpreter, which ran each line individually (though with the context of the previous lines).

IDEs such as Pyzo, PyCharm, Visual Studio Code, or even text editors are also recommended in writing longer programs.

Print Displays Data

Code run from a file doesn't show the evaluated result of every line (unlike code run from the interpreter). If we want to display a result, we need to use the command `print`.

`print` takes an input expression between parentheses, evaluates the expression, and displays the evaluated result in the interpreter.

For example:

`print(4 - 2)` displays `2` in the interpreter.

`print("53-705")` displays `53-705`; note that the quotes aren't included.

Printing Multiple Values

If you want to display multiple values in the interpreter on the same line, you have two choices.

First, if you're printing strings, you can concatenate them together.

```
print("Result: " + "2")
```

Alternatively, you can use commas to separate the values. `print` will then separate the printed values with spaces automatically. This is helpful for printing mixed types.

```
print("Result:", 2)
```

Comments are Ignored by the Computer

When writing a program with multiple lines, you might want to leave notes to yourself outside of the program commands. Use comments to do this.

Any text that follows a `#` on a line will be ignored by the computer:

```
print("Hello World") # a greeting
```

To comment out a block of code, put `"""` or `'''` at the beginning and end:

```
'''  
print("ignore")  
print("this")  
'''
```

You can also select a block of code and click Comment/Uncomment in IDEs to toggle comments.

Error Messages

Syntax Needs to be Exact

Computers aren't very clever. If you change the syntax of code even a little bit, the computer might not understand what you mean and will raise an error.

```
Print("Hello World") # NameError
```

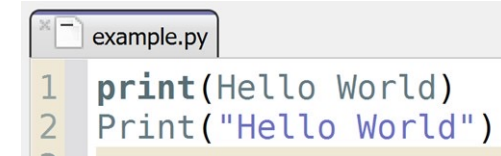
```
print "Hello World" # SyntaxError
```

When you get an error message, read it carefully. Error messages contain useful information that will help you fix your code.

Debug Errors By Reading the Message

1. Look for the line number. This line tells you approximately where the error occurred.
2. Look at the error type.
3. If it says `SyntaxError`, look for the inline arrow. The position gives you more information about the location of the problem (though it isn't always right).
4. If it says something else, read the error message. The error type and its message gives you information about what went wrong.

We'll talk more about the debugging process in future lectures.



```
example.py
1 print(Hello World)
2 Print("Hello World")
```

```
Running script: "C:\Users\river\Downloads\example.py"
File "C:\Users\river\Downloads\example.py", line 1
    print(Hello World)
          ^
SyntaxError: invalid syntax
>>>
```

inline arrow ↑
line number



```
example.py
1 print("Hello World")
2 Print("Hello World")
```

```
Running script: "C:\Users\river\Downloads\example.py"
Hello World
Traceback (most recent call last):
  File "C:\Users\river\Downloads\example.py", line 2, in
    <module>
      Print("Hello World")
NameError: name 'Print' is not defined
>>>
```

error type ↑

You Do: Debug the Code

Let's practice debugging! Given the following code and error message, determine A) what the problem is, and B) how to fix it.

Print('Your zip code is' + 53-703)

```
>>> print("Your zip code is " + 53-703)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>>
```


Whitespace is Syntax, Sometimes

Be careful when using whitespace (spaces, tabs, and the return key) – it can sometimes count as syntax too!

In general, whitespace at the beginning of a line has meaning; we'll discuss what it means more in a few weeks. Whitespace in the middle of tokens causes errors. Whitespace between tokens is okay.

```
print("Hello World") # IndentationError  
print ( "Hello World" ) # this is okay!
```

Variables

Variables Let Us Store Data

Our last core building block is the variable. Variables let us save data so we can reuse it in future computations.

A variable is a name that we define in the program (without quotes), like `x` or `result`. We define a variable with an equal sign:

```
variable = expression
```

Note that the variable can only go on the left side of this code, and its value (or an expression that evaluates to a value) goes on the right. For example:

```
myPet = "Stella"
```

```
result = 5 + 2
```

```
42 = foo # SyntaxError
```

Rules for Variable Names

Variable names can use any combination of uppercase letters, lowercase letters, digits, and underscores. They must start with a letter or `_`. Starting with a lowercase letter is recommended.

Variable names are case sensitive. For example, `Banana` is not the same as `banana`.

Mistyping a variable name is a common cause of `NameErrors`.

Expressions vs. Statements

Unlike everything else we've seen so far, a variable assignment is a statement, not an expression.

Recall that an expression is a piece of Python code that evaluates to a single data value. Data, operations, and the print command are all expressions. Variables are too!

A statement is an action taken by the program. It does not evaluate to a value; instead, it executes a change, then moves on to the next line. Variable assignments are statements.

Using and Updating Variables

Once we've defined a variable, we can use it in later expressions. Unlike in math, we can also change the variable to a new value, if needed.

```
x = 5
```

```
y = x - 2
```

```
x = x - 1
```

```
print("x:", x) # x: 4
```

Python is Sequential

Note that Python runs every line in order and doesn't peek ahead. If you want to use a variable, you must define it before it is used.

```
print(foo) # this causes an error!  
foo = 42
```

```
foo = 42  
print(foo) # this is fine!
```

Activity: Trace the Variable Values

Trace through the following lines of code. What values do **a** and **b** hold at the end?

a = 4

b = **a** - 2

a = **a** + 1

b = 7