Programming Basics

15-110 – Wednesday 09/02

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 Pyzo

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.

An IDE is a Text Editor for Programs

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

In this class, we recommend that you use the Pyzo IDE. It is fairly lightweight, which makes it good for novices.

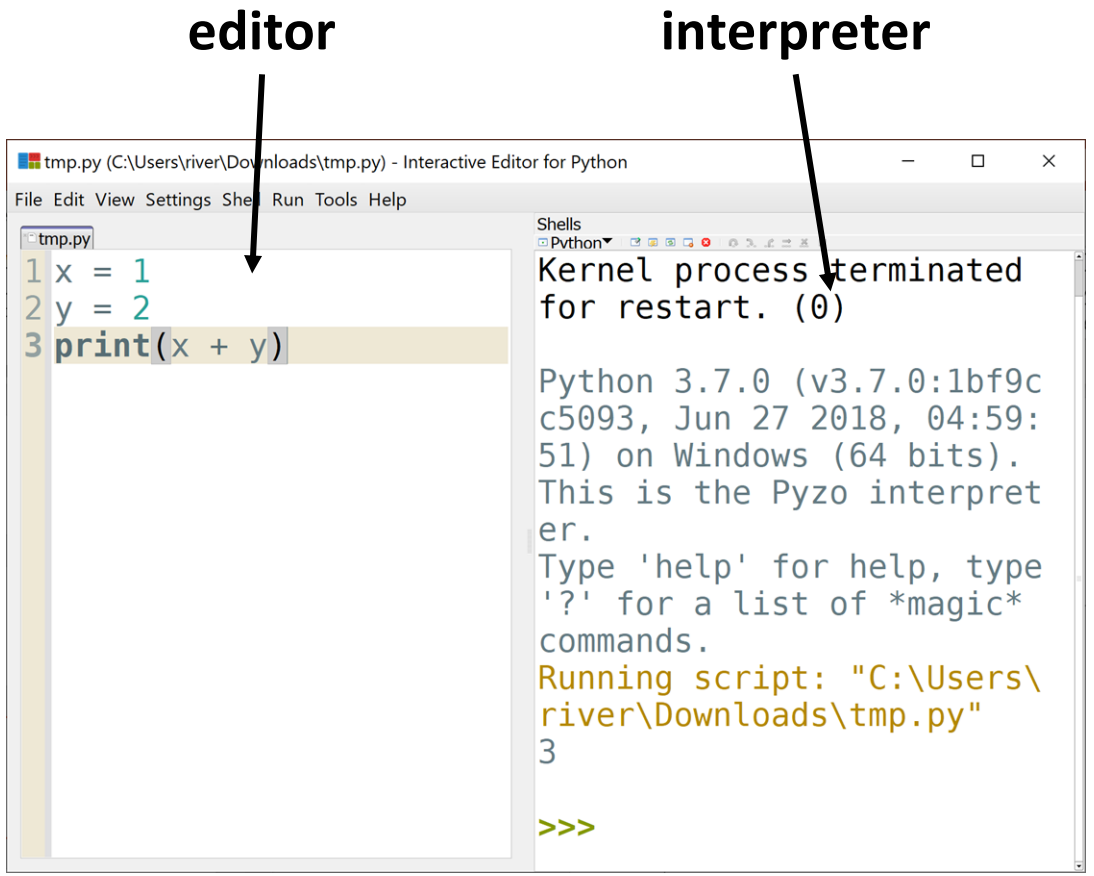
We will mostly use two parts of the Pyzo IDE while writing code- the **editor** and the **interpreter**.

Write in the Editor, Run in the Interpreter

The **editor** is just a normal text editor. When we save text, it is saved to a `.py` file, but this is still just normal text.

The **interpreter** does the actual work of converting our Python text into instructions the computer can run. This happens when you click 'Run File as Script'.

We can also run single lines of code in the interpreter directly. We'll start by doing that. In general, use the interpreter to run short tasks, and the editor for long tasks.



Sidebar: How Files Work

Your computer uses file and folders to organize data content locally (on the hardware).

A **file** is a single piece of content – a document, or a picture, or a song, or Python code.

A **folder** is a structure that holds files and/or other folders. Folders can be nested for further organization. Folders let you manage files directly.

You'll create many files (mostly .pdf and .py files) for this class. We recommend that you make a 15-110 folder to hold all your work.

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**.

Advanced Math Operations

There are two additional math operations that might not be as familiar to you.

Floor division, or **div** (//) divides numbers by rounding down to nearest whole number. This effectively cuts off any digits after the decimal point.

For example, `7 // 4` is equal to 1, not 1.75.

Modulo, or **mod** (%) finds the remainder when one number is divided by another.

For example, `7 % 4` is equal to 3.

Div and Mod are primarily used as a **floor operator** and to **repeat values**.

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"`.

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`.

Similarly, `"Hello" - "World"` results in a `TypeError`.

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 comparison operators are `<`, `>`, `<=`, or `>=`. Note that the two sides should be of the same type!

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

E.g., `"Hello" == "World"` evaluates to `False`

Names of Types in Python

In addition to the built-in operations, there are also built-in commands in Python called **functions**. These are algorithms that have already been added into the language. When we run a function on an input, it evaluates to an output.

To **call** a function, write the function's name followed by parentheses, with the **input** inside. We'll go over this in more depth in a future lecture.

`type(x)` is a built-in function that determines the type of the object **x**. The type could be:

int - integer

float - floating point number

str - string

bool - Boolean

For example:

`type(5)` evaluates to `int`

`type(5/3)` evaluates to `float`

`type("15-110")` evaluates to `str`

`type(True)` evaluates to `bool`

Type-casting Changes Types

We can also use these type names as functions, to change a value from one type to another.

For example, `str(42)` evaluates to `"42"`, and `int(3.14)` evaluates to `3`.

Type-casting is mainly useful for converting values to and from strings.

Activity: Predict the Type

Let's do a Kahoot to see if you can identify data types correctly!

Join by going to kahoot.it, then enter the game's pin.

If you're reading these slides after lecture, you can check out an asynchronous version of the Kahoot here:

https://kahoot.it/challenge/01795154?challenge-id=a750a494-3baa-4c36-81d2-898b6309e430_1599075659363

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**. Write lines of code in the editor, save, then click 'Run File as Script'.

Code run from a file doesn't show the evaluated result of every line. If we want to display a result, we need to use the **print function**. `print` just takes the input and displays the evaluated result in the interpreter.

For example:

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

`print("15-110")` displays `15-110`; 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, you can convert all the values to strings and concatenate them together.

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

Alternatively, you can use commas to separate the values. `print` will then separate the printed values with spaces automatically.

```
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 Pyzo to toggle comments.

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
```

We'll talk about errors much more in an upcoming lecture. For now, 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.

example.py

```
1  print("You're in " + 15-110)
```

```
Running script: "C:\Users\river\Downloads\example.py"
Traceback (most recent call last):
  File "C:\Users\river\Downloads\example.py", line 1, in
<module>
    print("You're in " + 15-110)
TypeError: can only concatenate str (not "int") to str

>>>
```

Sidebar: 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. And whitespace in the **middle of tokens** causes errors. But whitespace **between tokens** is okay.

```
    print("Hello World") # IndentationError
p r i n t ( "Hello World" ) # SyntaxError
print ( "Hello World" ) # this is okay!
```

Also, to save yourself trouble later on: in Pyzo, go to **File > Indentation**, and select **Use Spaces**, now.

Variables

Variables Let Us Store Data

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

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

variable = value

Note that the variable can only go on the left side of this code. For example:

```
x = 5 + 2
dog = "Stella"
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 `NameError`s.

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 evaluates to 5
x = x - 1 # x is 5 on the right, then changes to 4
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 have to 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
```

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
- **Feedback form: https://bit.ly/110-feedback**