Programming Basics

15-110 – 01/15

Announcement

- Check1 is due Monday at noon on Gradescope.
- Hw1 is due the following Monday at noon on Gradescope.

- Bring a laptop to recitation tomorrow if you have one.

Learning Objectives

- Recognize the basic building blocks of programming and use them in reading and writing Python programs.
  - Data types
  - Built-in functions
  - Variables

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, machine learning, and more.

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

Data holds Information

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 (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

A term like `4 ** 2` or `(5 - 2) / 3` is called an **expression;** that's 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 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, which 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 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 use `True` and `False` directly, or do a **comparison**. The comparison operators are `<`, `>`, `<=`, or `>=`. **Note:** The two sides must be 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.

`type(value)` is a built-in function that determines the type of the given value. This can be **int** (integer), **float** (floating point number), **str** (string), or **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

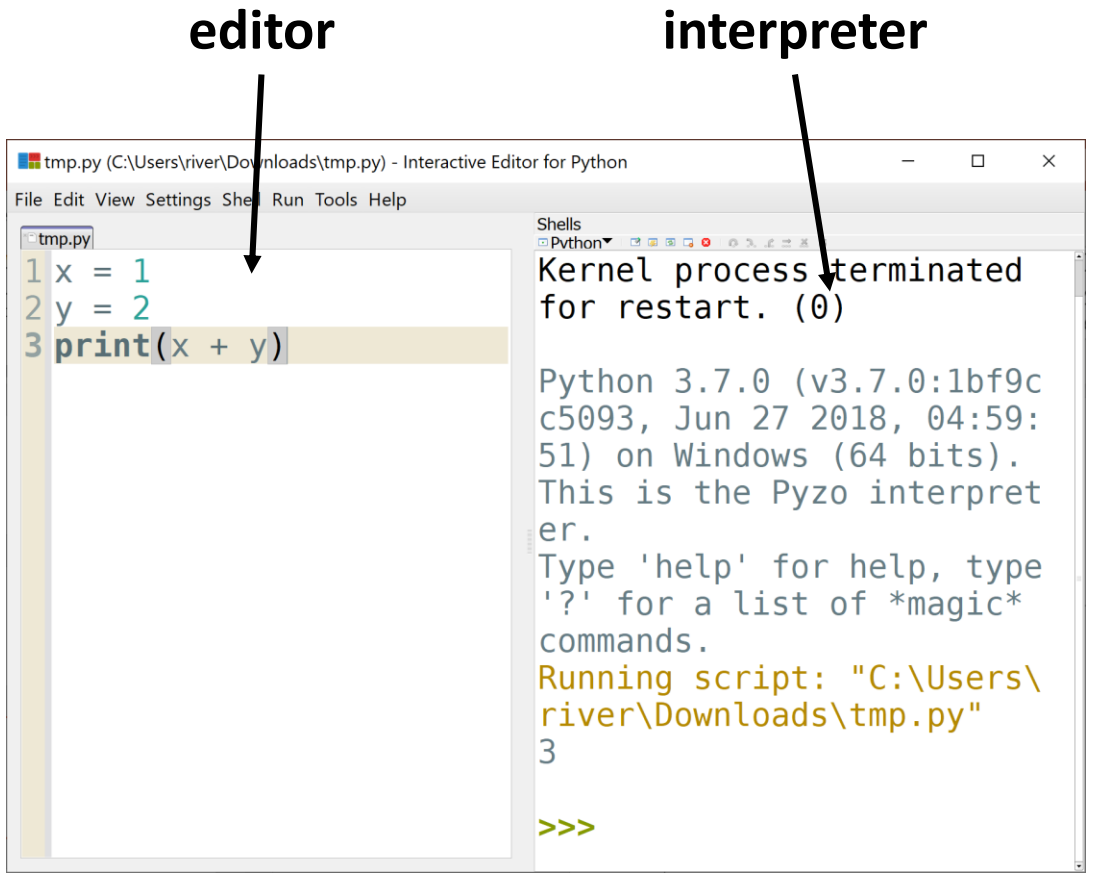
Let's do a quiz to see if you can identify data types correctly!

Interpreter for Short Tasks, Files for Long

Up until now, we've used the **interpreter** to test out bits of code. The interpreter evaluates a single line of code, then shows the result.

For longer programs, we'll want to write code in a **file**, where we can save it and run all of the lines together as often as we want.

Python file names should end in .py.



The `print` Function Displays Results

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`; 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")
"""
```

Sidebar: 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 result in an error.

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

We'll talk about errors much more in the next lecture. For now, try comparing your code against the lecture example code to find the difference.

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** has meaning; we'll discuss what it means more in a few weeks. But whitespace in the **middle or end** of a line 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 Let Us Store Data

Our last core building block is the **variable**. Variables let us **save data** so we can re-use them 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
```

Variables Name Rules

Variable names must start with a letter or _. Starting with a lower case letter is recommended.

Variable names can use any combination of uppercase letters, lowercase letters, digits, and underscores.

Variables are case sensitive. For example `Value` is not the same as `value`.

Mistyping a variable name is a common cause of NameErrors.

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

Python is Sequential

Note that Python runs every line in order, but 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!
```

Learning Objectives

- Recognize the basic building blocks of programming and use them in reading and writing Python programs
  - Data types
  - Built-in functions
  - Variables