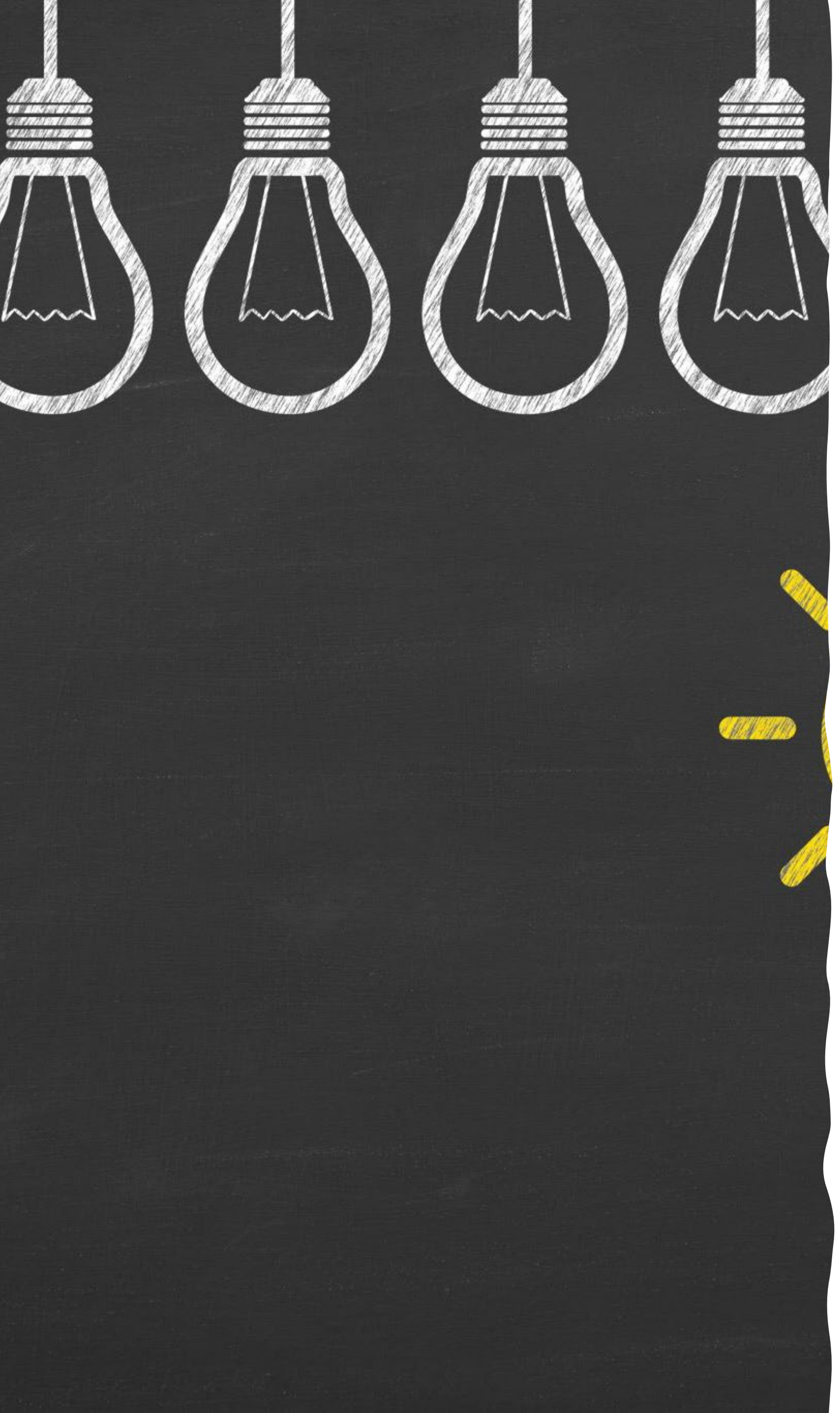


CMSC 105 Elementary Programming

Acknowledgement: These slides are adapted from slides provided with “Introduction to Programming Using Python, Liang (Pearson 2013)” and slides shared by Dr. Jory Denny

Outline

Introduction to Programming

Walk through Programming

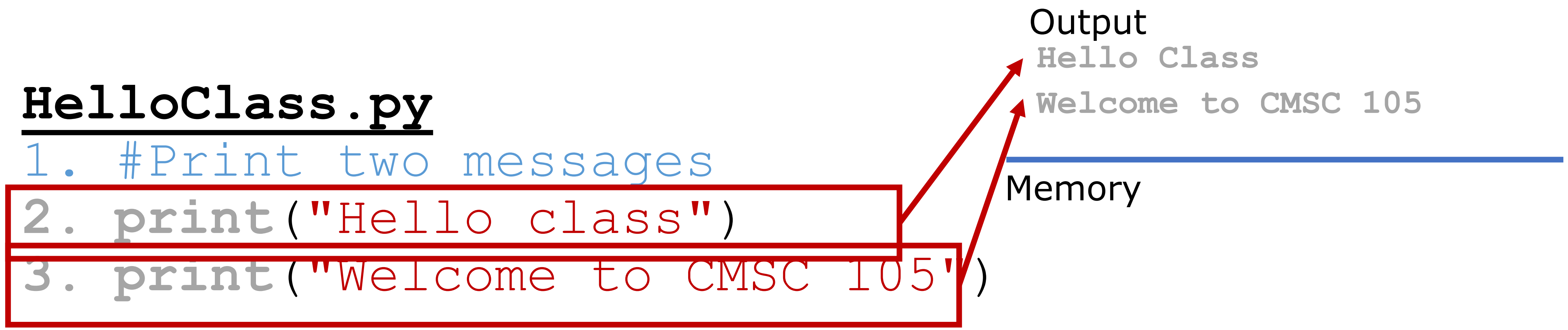
Hello Class!

HelloClass.py

```python
#Print two messages
print("Hello Class!")
print("Welcome to CMSC 105")
```

- **Run:** `python3 HelloClass.py`

Tracing


HelloClass.py

```
#Print two messages
print("Hello class")
print("Welcome to CMSC 105")
```

```
Hello Class
Welcome to CMSC 105
```

- Tracing is the activity of following a computation by hand.
- Not a classroom activity! Professionals do this regularly on sections of programs
  - Typically to determine when something goes wrong

Anatomy of a Python Program

- Statements
- Comments
- Indentation

Statement

- A **statement** represents an action or a sequence of actions.
- The statement **print**("Hello class") in the program is a statement to display the message "Hello class".

HelloWorld.py

```python
#Print two messages
print("Hello class")
print("Welcome to CMSC 105")
```

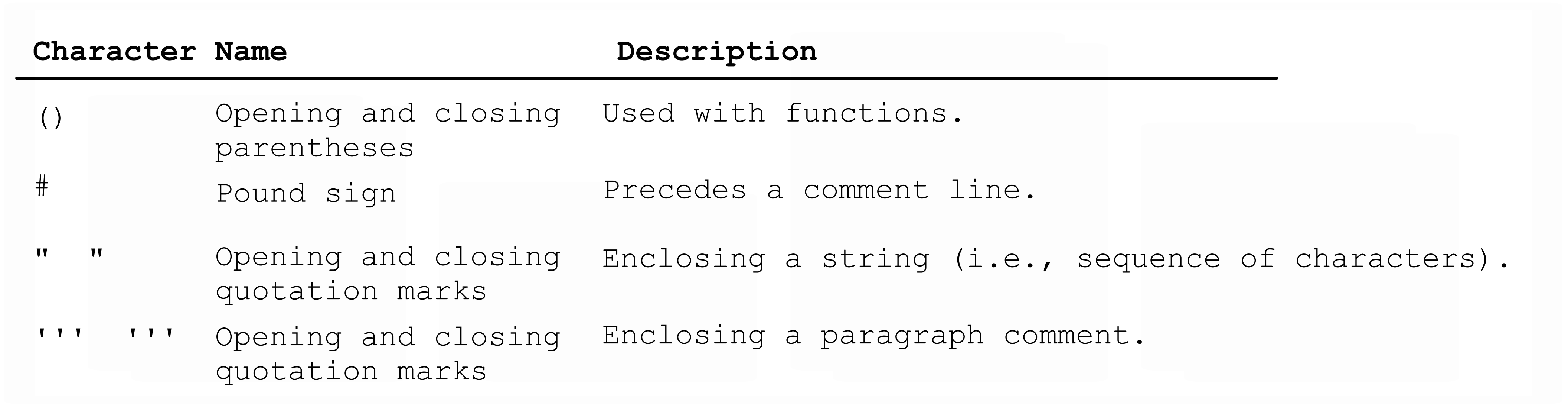
Indentation

- The indentation matters in Python. Note that the statements are entered from the first column in the new line. It would cause an error if the program is typed as follows, for example:

HelloWorld.py

```python
#Print two messages
 print("Hello class")
print("Welcome to CMSC 105")
```

Special Symbols



Character	Name	Description
()	Opening and closing parentheses	Used with functions.
#	Pound sign	Precedes a comment line.
" "	Opening and closing quotation marks	Enclosing a string (i.e., sequence of characters).
''' '''	Opening and closing quotation marks	Enclosing a paragraph comment.

Reserved words

- **Reserved words** or **keywords** are words that have a specific meaning to the compiler and cannot be used for other purposes in the program. We will see many of these during the course of the semester. The previous program doesn't have any specifically.

Programming Style and Documentation

- **Appropriate Comments**
- **Naming Conventions**
- **Proper Indentation and Spacing Lines**

Appropriate Comments

- Include a summary at the beginning of the program to explain what the program does, its key features, its supporting data structures, and any unique techniques it uses.
- Document each variable, function, and class
- Include your name and a brief description at the beginning of the program.

Naming Conventions

- Choose meaningful and descriptive names.

Proper Indentation and Spacing

- **Indentation**
  - Indent two spaces.
  - A consistent spacing style makes programs clear and easy to read, debug, and maintain.

- **Spacing**
  - Use blank line to separate segments of the code.

Programming Errors

- **Syntax Errors**
  - Error in writing python syntax
- **Runtime Errors also called Exceptions**
  - Causes the program to abort
- **Logic Errors**
  - Produces incorrect result

Syntax Errors

- **Syntax errors** are errors from incorrectly written Python code.
- Anatomy of a compiler error:
  File "filename.py", line num
  ErrorType: Confusing description of error including code where it occurs.
- Deal with errors by experience, google, stack overflow, etc.
  After you have exhausted these resources…piazza/ask me.
  Advice, always handle the first error…not the last one.

HelloWorld.py

```
//Print two messages
 print("Hello class"
print(Welcome to CMSC 105")
```

Can anyone spot the syntax errors?

Runtime Errors

- Runtime errors occur from impossible commands encountered while executing the program
- Error message shows a "traceback" of the program execution. Right now, just know that this tells where/why the error occurs.

HelloWorld.py

```
print(1/0)
```

Logic Errors

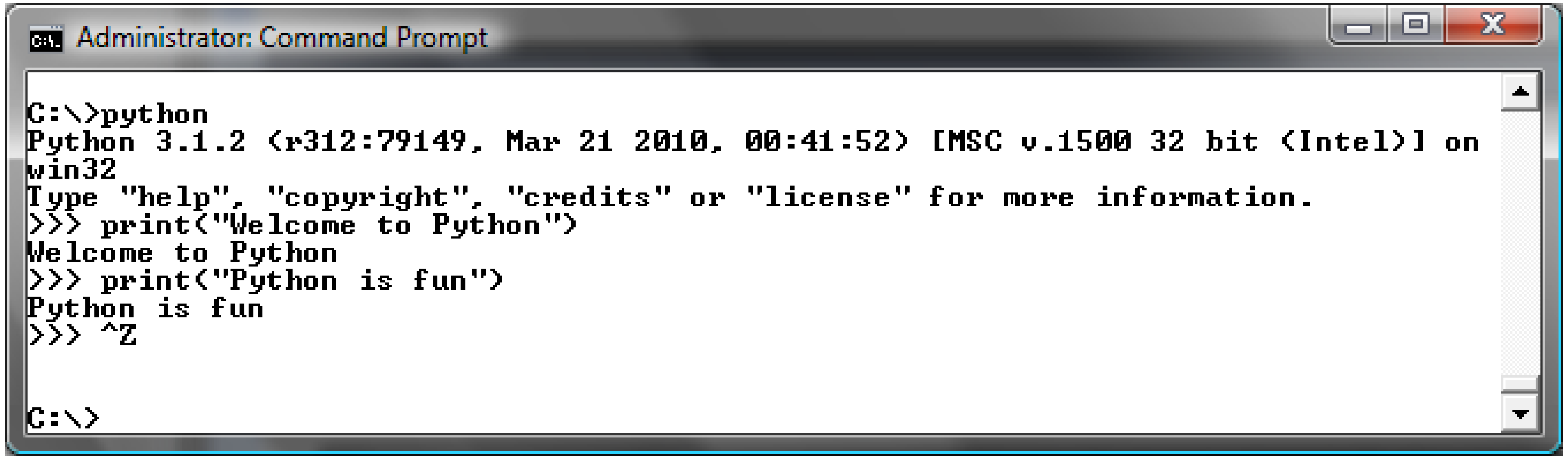
- Logic errors are incorrect computations that run without exceptions but produce the incorrect output

HelloWorld.py

```
#Celcius conversion
print("Celcius 35 is Fahrenheit",
 (9//5)*35+32)
```

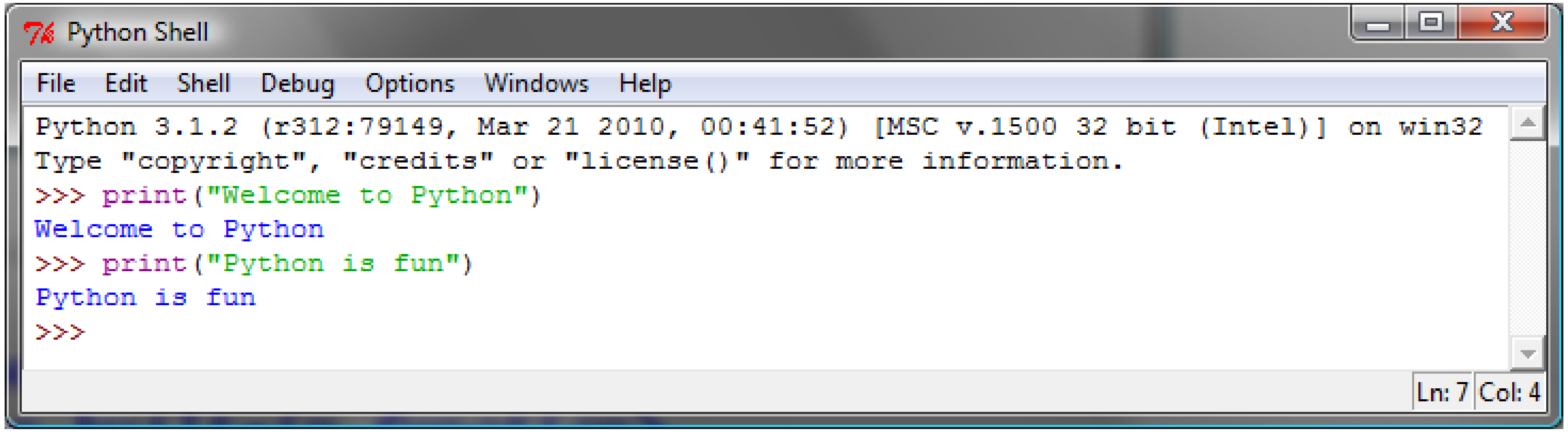
Can anyone spot the logic error?

Launch Python



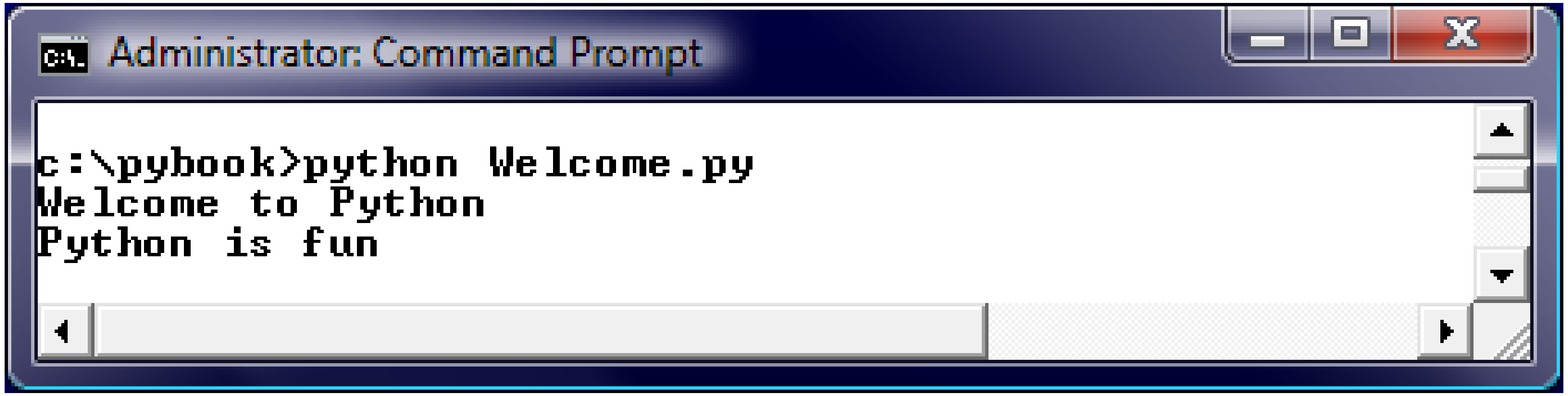
```
C:\>python
Python 3.1.2 (r312:79149, Mar 21 2010, 00:41:52) [MSC v.1500 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Welcome to Python")
Welcome to Python
>>> print("Python is fun")
Python is fun
>>> ^Z


C:\>
```

Launch Python IDLE



```
Python 3.1.2 (r312:79149, Mar 21 2010, 00:41:52) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("Welcome to Python")
Welcome to Python
>>> print("Python is fun")
Python is fun
>>>
```

Run Python Script



```
c:\pybook>python Welcome.py
Welcome to Python
Python is fun
```

A Simple Python Program

```
# Display two messages
print("Welcome to Python")
print("Python is fun")
```

Welcome

Note: Clicking the green button displays the source code with interactive animation. You can also run the code in a browser. Internet connection is needed for this button.

Trace a Program Execution

Execute a statement

```
# Display two messages
print("Welcome to Python")
print("Python is fun")
```

Trace a Program Execution

Execute a statement

```
# Display two messages
print("Welcome to Python")
print("Python is fun")
```

Python IDE

- IDE stands for Integrated Development Environment. It’s a coding tool which allows you to write, test, and debug your code in an easier way
- Some possible Integrated Development Environment platforms for Python can be found at:

https://wiki.python.org/moin/IntegratedDevelopmentEnvironments

Reading input from the console

ComputeArea.py

```
# Assign a value to radius
radius = eval(input("Enter a value for a radius: "))

# Compute the area
area = radius * radius * 3.14159

# Display the result
print("The area for a circle with radius",
          radius, "is", area)
```

input is a function to collect key strokes from the console

eval is a function that converts those key strokes to a value

Data Types

- Overall, data types define the available operations on and range of the data representation. Additionally, it notes how it is stored in memory.

- Right now we have seen:
  - **Strings** – sequences of characters, e.g., "Hello"
  - **Floating point numbers** – representing real numbers with fractional components, e.g., 3.54
  - **Integers** – representing positive and negative whole numbers, e.g., 15

Variables and Naming

Identifiers (names)

- Identifiers are the names that **identify** the elements such as variables and functions in a program.
- An **identifier** is a sequence of characters that consist of letters, digits, underscores (_), and asterisk (*).
- An identifier must start with a letter or an underscore (_).
- An identifier cannot be a reserved word. (See Appendix A, "Python Keywords," for a list of reserved words).
- An identifier can be of any length.

Which of these are invalid identifiers?
Area, 2volume, miles, radius, if,

Literals

- A **literal** is a constant value that appears directly in the program. For example, 34, 1,000,000, and 5.0 are literals in the following statements:

```
i = 34
x = 1000000
d = 5.0
```

Variables

- A **variable** is a named piece of data (memory). It stores a **value**!
- Variables are used to reference values that may be changed in the program
- It has a **type** that defines how the memory is interpreted and what operations are allowed

```
      var = value
```

```
Example: radius = 5
```

Expressions

- **Expressions** are combinations of literals, variables, operations, and function calls that generate new values

$$\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

is translated to

```
(3+4*x)/5 - 10*(y-5)*(a+b+c)/x + 9*(4/x +
(9+x)/y)
```

Assignment Statements

- Assignment statements give values to a variable

```
x = 1;              // Assign 1 to x;
radius = 1.0;       // Assign 1.0 to radius;
a = 'A';            // Assign 'A' to a;
```

Simultaneous assignment

- Python allows a shorthand to create/assign multiple variables at a time. Variables and expressions will be comma separated. An example:

  `Examples:`

  - `x, y = (a+b)/2, (a-b)/2`
  - `number1, number2 = eval(input(“Enter 2 numbers separated by commas:”)`

Named Constants

- Often, we need constants in programs, e.g., π., whose value never changes.
- Python does not have a special syntax for naming constants. You can simply create a variable to denote a constant. To distinguish a constant from a variable, use all uppercase letters to name a constant.

`Examples:`

- `PI = 3.14159`
- `SIZE = 3`

Naming Conventions

- Choose meaningful and descriptive names.
- Typically begin with lower case
- Python typically names with underscores separating words (snake casing), but other styles capitalize the first letter of each subsequent word (camel casing):
  - my_area_variable
  - myAreaVariable
- Constants will be all caps using snake casing: MY_PI_CONSTANT
- Be consistent!

Scientific Notation

- Floating-point literals can also be specified in scientific notation

- Example,
  - 1.23456e+2, same as 1.23456e2, is equivalent to 123.456
  - 1.23456e-2 is equivalent to 0.0123456. E (or e) represents an exponent and it can be either in lowercase or uppercase.

Expressions

Numeric Operators

Name	**Meaning**	**Example**	**Result**
+	Addition	34 + 1	35
-	Subtraction	34.0 – 0.1	33.9
*	Multiplication	300 * 30	9000
/	Float Division	1 / 2	0.5
//	Integer Division	1 // 2	0
**	Exponentiation	4 ** 0.5	2.0
%	Remainder	20 % 3	2

Numeric Operators cont’d

- The / operator performs float division:

  Example:

  4/2 yields 2.0, 2/4 yields 0.5

- The // operator performs an integer division:

  Example:

  5//2 yields 2, 2//4 yields 0

- To compute a^b, you can write a ** b
- The % operator (known as remainder or modulo operator) yields the remainder after division

  Example:

  26 % 8 yields 2

Question

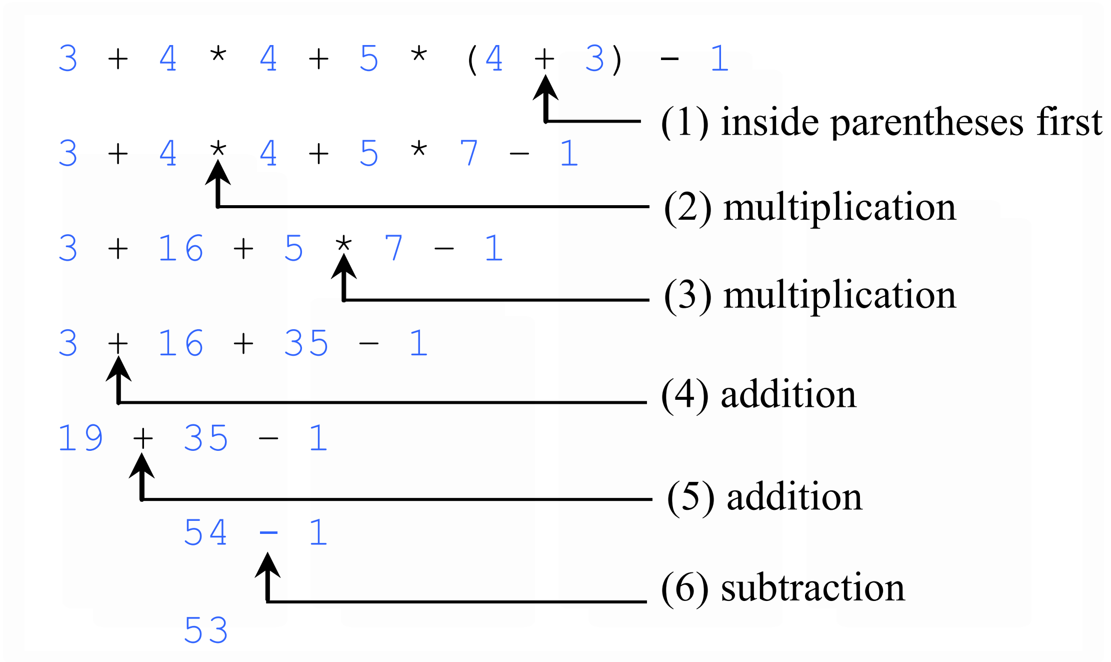
- If today is Thursday and you are planning to meet a friend after 10 days. What day is in 10 days?

Note: Assume Sunday is day 0 of the week

How to Evaluate an Expression

- Though Python has its own way to evaluate an expression behind the scene, the result of a Python expression and its corresponding arithmetic expression are the same.

- Therefore, you can safely apply the arithmetic rules for evaluating a Python expression.



Underflow and Overflow

- When a floating-point variable is assigned a value that is too large (in size) to be stored, it causes overflow, a run time exception.

  Example:
  245 ** 1000

- When a floating-point number is too small (i.e., too close to zero) to be stored, it causes underflow. Python approximates it to zero. So normally you should not be concerned with underflow.

Augmented Assignment Operators

Operator	Name	Example	Equivalent
`+=`	Addition assignment	`i += 8`	`i = i + 8`
`-=`	Subtraction assignment	`i -= 8`	`i = i - 8`
`*=`	Multiplication assignment	`i *= 8`	`i = i * 8`
`/=`	Float division assignment	`i /= 8`	`i = i / 8`
`//=`	Integer division assignment	`i //= 8`	`i = i // 8`
`%=`	Remainder assignment	`i %= 8`	`i = i % 8`
`**=`	Exponent assignmnet	`i **= 8`	`i = i ** 8`

Type Conversion

Type Conversion

- Use **`int()`**, **`float()`**, **`str()`** to convert any data type to integer, floating-point, or string respectively
- Consider the following statements and their results:
  - `int(4.7)` → 4
  - `float(4)` → 4.0
  - `str(4)` → "4"
- To round floating point numbers use round()
  - `round(4.7)` → 5

Next Topic

Simple Sequential Programs

Thank you!
Questions?