# **Introduction to Python – I**

The purpose of this lab is to familiarize you with this term's lab system and to diagnose your programming ability and facility with Python. This course uses Python for all of its labs, and you will be called on to understand the functioning of large systems, as well as to write significant pieces of code yourself. While coding is not, in itself, a focus of this class, artificial intelligence is a hard subject full of subtleties. As such, it is important that you be able to focus on the problems you are solving, rather than the mechanical code necessary to implement the solution.

Python resources

Udacity offers a really helpful Introductory course for Python Beginners <https://classroom.udacity.com/courses/ud1110>

**Python:** There are a number of versions of Python available. This course will use standard Python from <http://www.python.org/>. If you are running Python on your own computer, you should download and install latest version (Currently Python 3.7.4) from <http://www.python.org/download/> .

## Essentials of a Python program:

In most of today’s written languages, words by themselves do not make sense unless they are in certain order and surrounded by correct punctuation symbols. This is also the case with the Python programming language. The Python interpreter is able to interpret and run correctly structured Python programs. For example, the following Python code is correctly structured and will run:



Many other languages require a lot more structure in their simplest programs, but in Python this single line, which prints a short message, is sufficient. A very informative example of Python’s syntax which does (almost) exactly the same thing:



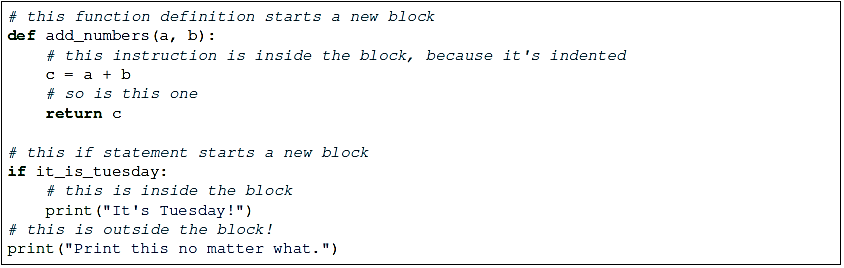
A hash (#) denotes the start of a comment. The interpreter will ignore everything that follows the hash until the end of the line.

### Flow of control

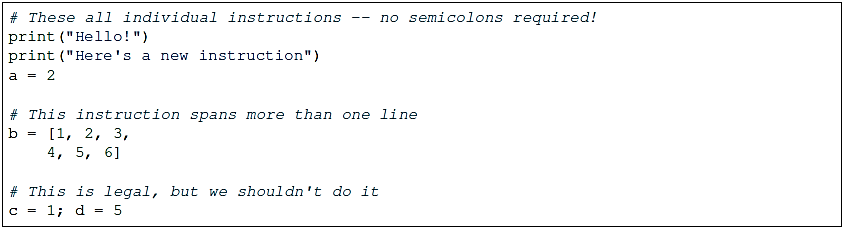
In Python, statements are written as a list, in the way that a person would write a list of things to do. The computer starts off by following the first instruction, then the next, in the order that they appear in the program. It only stops executing the program after the last instruction is completed. We refer to the order in which the computer executes instructions as the flow of control. When the computer is executing a particular instruction, we can say that control is at that instruction.

### Indentation and (lack of) semicolons

Many languages arrange code into blocks using curly braces **({** and **})** or BEGIN and END statements – these languages encourage us to indent blocks to make code easier to read, but indentation is not compulsory. Python uses indentation only to delimit blocks, so we must indent our code:



In many languages we need to use a special character to mark the end of each instruction – usually a semicolon. Python uses ends of lines to determine where instructions end (except in some special cases when the last symbol on the line lets Python know that the instruction will span multiple lines). We may optionally use semicolons – this is something we might want to do if we want to put more than one instruction on a line (but that is usually bad style):



### Built-in types

There are many kinds of information that a computer can process, like numbers and characters. In Python, the kinds of information the language is able to handle are known as types. Many common types are built into Python – for example integers, floating-point numbers and strings. Users can also define their own types using classes. In many languages a distinction is made between built-in types (which are often called “primitive types” for this reason) and classes, but in Python they are indistinguishable. Everything in Python is an object (i.e. an instance of some class) – that even includes lists and functions.

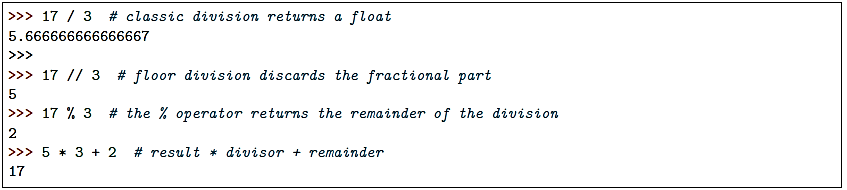
Python is a dynamically (and not statically) typed language. That means that we don’t have to specify a type for a variable when we create it – we can use the same variable to store values of different types. However, Python is also strongly typed – at any given time, a variable has a definite type. If we try to perform operations on variables which have incompatible types (for example, if we try to add a number to a string), Python will exit with a type error instead of trying to guess what we mean.

Python has large collection of built-in functions that operate on different kinds of data to produce all kinds of results. One function is called type, and it returns the type of any object.



#### Numbers:

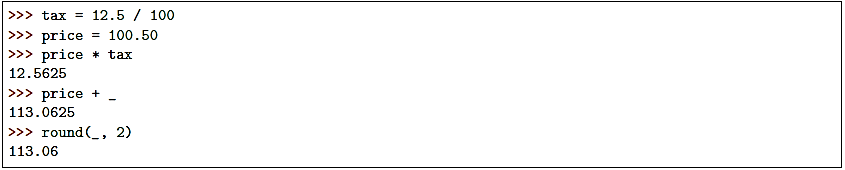
The interpreter acts as a simple calculator: you can type an expression at it and it will write the value. Expression syntax is straightforward: the operators +, -, \* and / work just like in most other languages.



The equal sign (=) is used to assign a value to a variable. Afterwards, no result is displayed before the next interactive prompt:

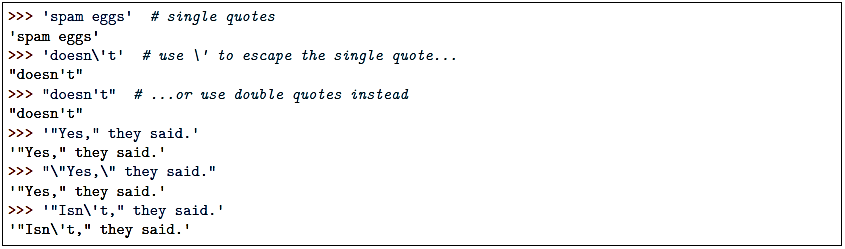


In interactive mode, the last printed expression is assigned to the variable \_. This means that when you are using Python as a desk calculator, it is somewhat easier to continue calculations, for example:

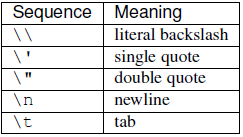


#### Strings

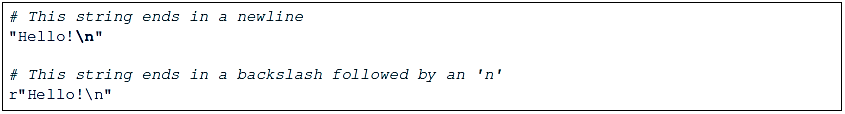
A string is a sequence of characters. Python can also manipulate strings, which can be expressed in several ways. They can be enclosed in single quotes ('...') or double quotes ("...") with the same result. \ can be used to escape quotes. In the interpreter, the output string is enclosed in quotes and special characters are escaped with backslashes.



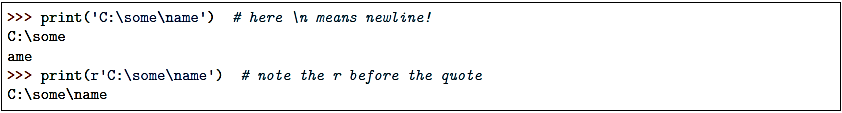
Some common escape sequences:



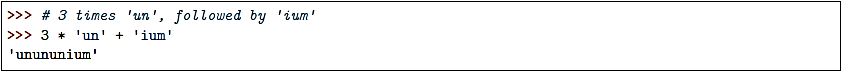
Sometimes we may need to define string literals which contain many backslashes – escaping all of them can be tedious. We can avoid this by using Python’s raw string notation. By adding an **r** before the opening quote of the string, we indicate that the contents of the string are exactly what we have written, and that backslashes have no special meaning. For example:



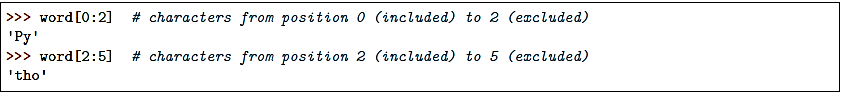
Consider another example:



Strings can be concatenated (glued together) with the + operator, and repeated with \*:



Strings can be *indexed* (subscripted), with the first character having index 0. There is no separate character type; a character is simply a string of size one:



### Files

Although the print function prints to the console by default, we can also use it to write to a file. Here is a simple example:



In the **with** statement the file **myfile.txt** is opened for writing and assigned to the variable **myfile**. Inside the **with** block, **Hello!** followed by a newline is written to the file. The **w** character passed to open indicates that the file should be opened for writing. The **with** statement automatically closes the file at the end of the block, even if an error occurs inside the block.

As an alternative to **print**, we can use a file’s **write** method as follows:



Unlike **print**, the **write** method does not add a newline to the string which is written.

We can read data from a file by opening it for reading and using the file’s **read** method:

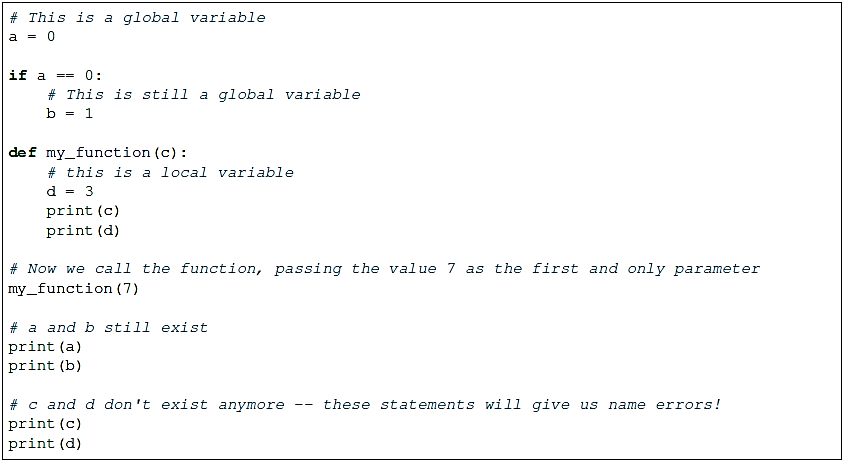


This reads the contents of the file into the variable data. Note that this time we have passed **r** to the open function. This indicates that the file should be opened for reading.

### Variable scope and lifetime

Where a variable is accessible and how long it exists depend on how it is defined. We call the part of a program where a variable is accessible its scope, and the duration for which the variable exists its lifetime. A variable which is defined in the main body of a file is called a global variable. It will be visible throughout the file, and also inside any file which imports that file. Global variables can have unintended consequences because of their wide-ranging effects – that is why we should almost never use them. Only objects which are intended to be used globally, like functions and classes, should be put in the global namespace. A variable which is defined inside a function is local to that function. It is accessible from the point at which it is defined until the end of the function, and exists for as long as the function is executing.

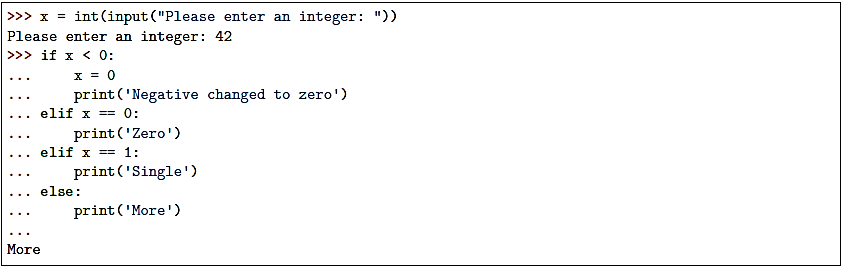
Here is an example of variables in different scopes:



## Selection control statements

### Selection: if statement

Perhaps the most well-known statement type is the if statement. For example:



There can be zero or more **elif** parts, and the else part is optional. The keyword ‘**elif**’ is short for ‘**else if’**, and is useful to avoid excessive indentation. An if … elif … elif … sequence is a substitute for the switch or case statements found in other languages.

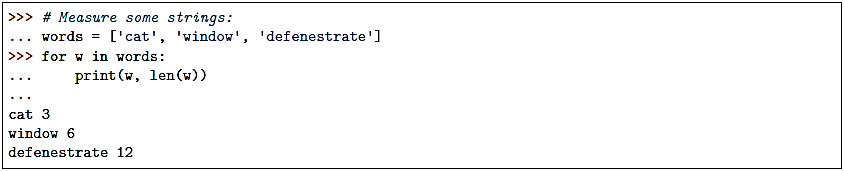
The interpreter will treat all the statements inside the indented block as one statement – it will process all the instruction in the block before moving on to the next instruction. This allows us to specify multiple instructions to be executed when the condition is met.

### The for Statement

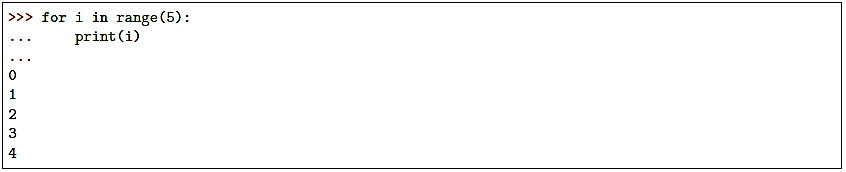
The **for** statement in Python differs a bit from what you may be used to in C. Rather than always giving the user the ability to define both the iteration step and halting condition (as C), Python’s for statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence. For example:



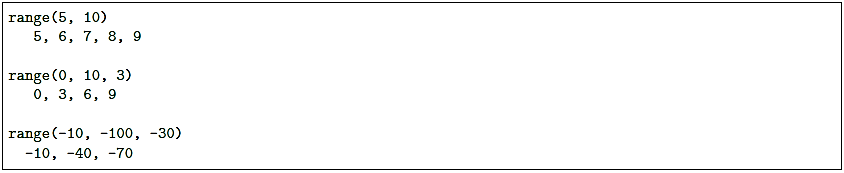
Consider another example.



If you do need to iterate over a sequence of numbers, the built-in function **range ()** comes in handy. It generates arithmetic progressions:



The given end point is never part of the generated sequence; range(10) generates 10 values, the legal indices for items of a sequence of length 10. It is possible to let the range start at another number, or to specify a different increment (even negative; sometimes this is called the ‘step’):



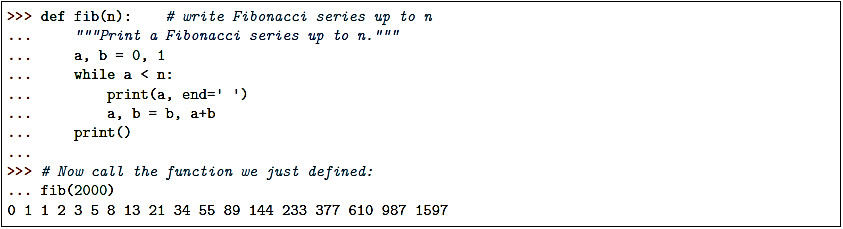
## Defining Functions

A function is a sequence of statements which performs some kind of task. Here is a definition of a simple function which takes no parameters and doesn’t return any values:



We use the **def** statement to indicate the start of a function definition. The next part of the definition is the function name, in this case print\_a\_message, followed by round brackets (the definitions of any parameters that the function takes will go in between them) and a colon. Thereafter, everything that is indented by one level is the body of the function.

We can create a function that writes the Fibonacci series to an arbitrary boundary:

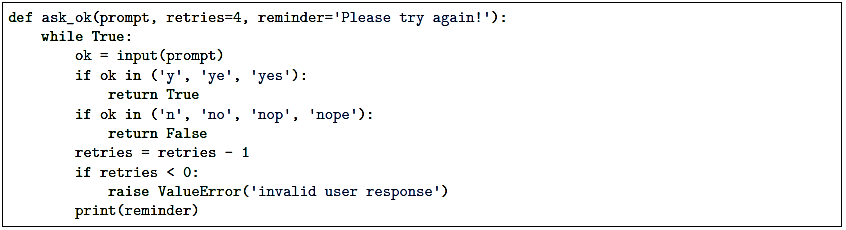


A function definition introduces the function name in the current symbol table. The value of the function name has a type that is recognized by the interpreter as a user-defined function. This value can be assigned to another name which can then also be used as a function. This serves as a general renaming mechanism:



### Default Argument Values

The most useful form is to specify a default value for one or more arguments. This creates a function that can be called with fewer arguments than it is defined to allow. For example:

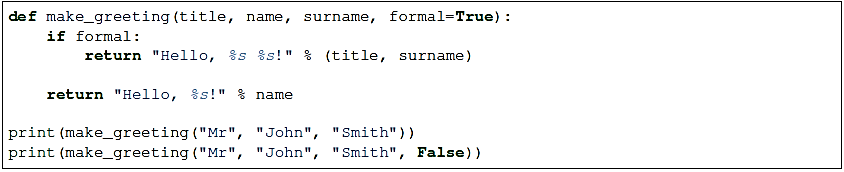


This function can be called in several ways:

* giving only the mandatory argument: ask\_ok('Do you really want to quit?')
* giving one of the optional arguments: ask\_ok('OK to overwrite the file?', 2)
* or even giving all arguments: ask\_ok('OK to overwrite the file?', 2, 'Come on, only yes or no!')

This example also introduces the **in** keyword. This tests whether or not a sequence contains a certain value.

In Python, there can only be one function with a particular name defined in the scope – if you define another function with the same name, you will overwrite the first function. You must call this function with the correct number of parameters, otherwise you will get an error. Sometimes there is a good reason to want to have two versions of the same function with different sets of parameters. You can achieve something similar to this by making some parameters optional. To make a parameter optional, we need to supply a default value for it. Optional parameters must come after all the required parameters in the function definition:



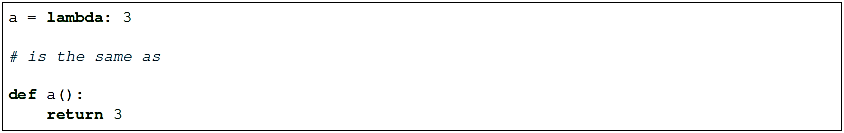
When we call the function, we can leave the optional parameter out – if we do, the default value will be used. If we include the parameter, our value will override the default value.

### Lambdas

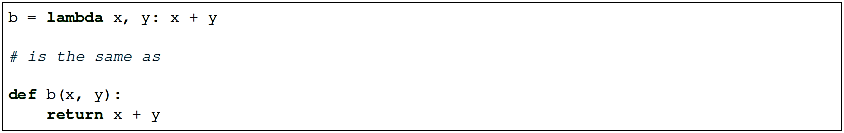
We have already seen that when we want to use a number or a string in our program we can either write it as a literal in the place where we want to use it or use a variable that we have already defined in our code. For example, print("Hello!") prints the literal string "Hello!", which we haven’t stored in a variable anywhere, but print(message) prints whatever string is stored in the variable message.

We have also seen that we can store a function in a variable, just like any other object, by referring to it by its name (but not calling it). Is there such a thing as a function literal? Can we define a function on the fly when we want to pass it as a parameter or assign it to a variable, just like we did with the string "Hello!"?

The answer is yes, but only for very simple functions. We can use the lambda keyword to define anonymous, one-line functions inline in our code:



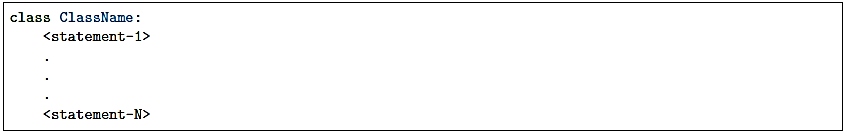
Lambdas can take parameters – they are written between the lambda keyword and the colon, without brackets. A lambda function may only contain a single expression, and the result of evaluating this expression is implicitly returned from the function (we don’t use the return keyword):



## Class Definitions:

A class serves as the primary means for abstraction in object-oriented programming. In Python, every piece of data is represented as an instance of some class. A class provides a set of behaviors in the form of member functions (also known as methods), with implementations that are common to all instances of that class. A class also serves as a blueprint for its instances, effectively determining the way that state information for each instance is represented in the form of attributes (also known as fields, instance variables, or data members).

The simplest form of class definition looks like this:



**Example: CreditCard Class**

As a first example, we will define a CreditCard class. The instances defined by the CreditCard class provide a simple model for traditional credit cards. They have identifying information about the customer, bank, account number, credit limit, and current balance. The class restricts charges that would cause a card’s balance to go over its spending limit, but it does not charge interest or late payments.

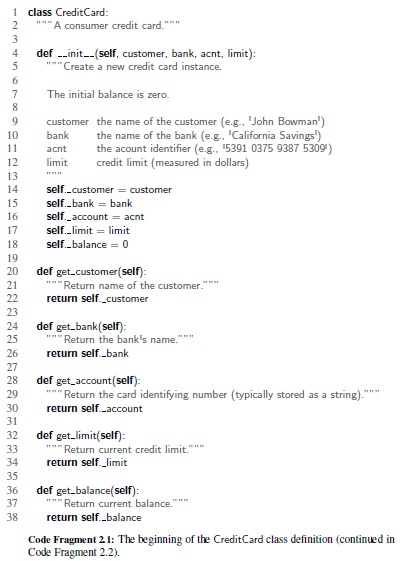
The construct begins with the keyword, class, followed by the name of the **class**, a colon, and then an indented block of code that serves as the body of the class. The body includes definitions for all methods of the class. These methods are defined as functions, yet with a special parameter, named **self**, that serves to identify the particular instance upon which a member is invoked.

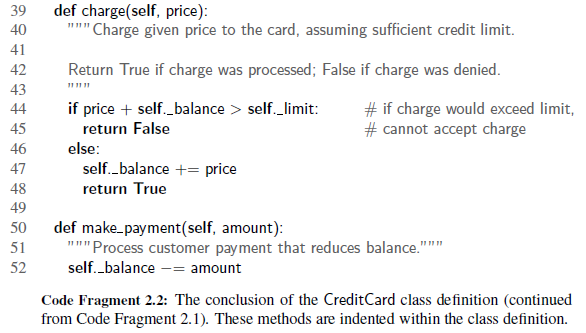
By convention, names of members of a class (both data members and member functions) that start with a single underscore character (e.g., \_secret) are assumed to be nonpublic and should not be relied upon.

### The self Identifier:

In Python, the self identifier plays a key role. In the context of the CreditCard class, there can presumably be many different CreditCard instances, and each must maintain its own balance, its own credit limit, and so on. Therefore, each instance stores its own instance variables to reflect its current state.

Syntactically, self identifies the instance upon which a method is invoked. For example, assume that a user of our class has a variable, my\_card, that identifies an instance of the CreditCard class. When the user calls my\_card.get\_balance( ), identifier self, within the definition of the get\_balance method, refers to the card known as my\_card by the caller. The expression, self.\_balance refers to an instance variable, named \_balance, stored as part of that particular credit card’s state.





### The Constructor

A user can create an instance of the CreditCard class using a syntax as:

cc = CreditCard( John Doe, 1st Bank , 5391 0375 9387 5309 , 1000)

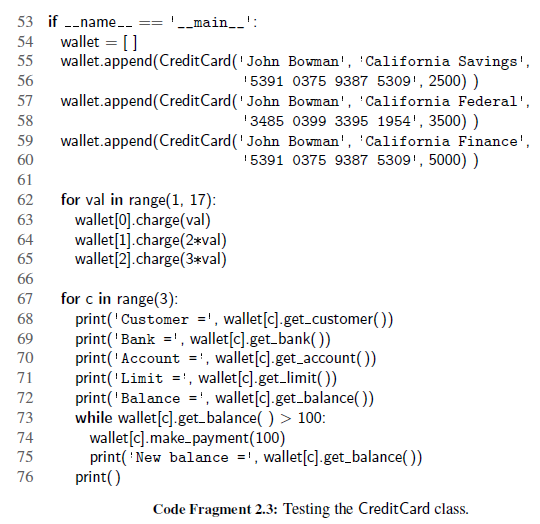
Internally, this results in a call to the specially named \_ \_init\_ \_ method that serves as the constructor of the class. Its primary responsibility is to establish the state of a newly created credit card object with appropriate instance variables. By the conventions, a single leading underscore in the name of a data member, such as \_balance, implies that it is intended as nonpublic. Users of a class should not directly access such members.

Many classes like to create objects with instances customized to a specific initial state. Therefore a class may define a special method named \_ \_init\_ \_(), like this:



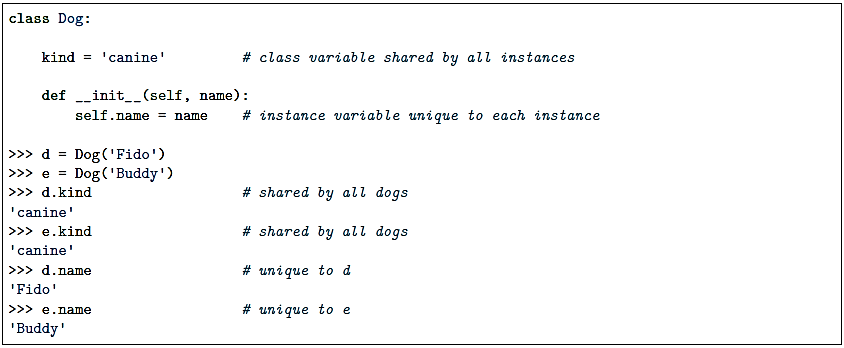
Testing the Class:

We will demonstrate some basic usage of the CreditCard class, inserting three cards into a list named wallet. We use loops to make some charges and payments, and use various accessors to print results to the console. These tests are enclosed within a conditional, if \_ \_name == ‘\_ \_main\_ \_’:, so that they can be embedded in the source code with the class definition.



### Class and Instance Variables

Generally speaking, instance variables are for data unique to each instance and class variables are for attributes and methods shared by all instances of the class:



## Lab Tasks

**Exercise 1.1**

**Type each of the following expressions into python3. What value do each of the following Python expressions evaluate to? Is that value an integer or a floating point?**

1. 250
2. 28 % 5
3. 2.5e2
4. 3e5
5. 3 \* 10\*\*5
6. 20 + 35 \* 2  
   *Why is this different from (20 + 35) \* 2?*
7. 2 / 3 \* 3
8. 2 // 3 \* 3  
   *Why is this different from 2 / 3 \* 3?*
9. 25 - 5 \* 2 - 9  
   *Is this different from ((25 - 5) \* 2) - 9 and/or 25 - ((5 \* 2) - 9)? Why?*

**Exercise 1.2**

Suppose we are making ice cream sundaes. We have four flavors of ice cream: vanilla, chocolate, strawberry, and pistacchio. And we have three sauces: caramel, butterscotch, and chocolate. How many different ice cream sundaes can we make? Define a function sundaes() to systematically print out every possible combination, one per line. For example, the first line should say "vanilla ice cream sundae with caramel sauce". You should create a list to hold each class of ingredient, and use nested for loops to iterate over these lists to generate the combinations. At the end, your function should return an integer giving the total number of combinations.

To get you started here a few things you will need:

flavors = ["vanilla", "chocolate", "strawberry", "pistacchio"]

sauces = ["caramel", "butterscotch", "chocolate"]

print(flavor + " ice cream sundae with " + sauce + " sauce")

**Exercise 1.3**

Consider the following output as shown:

1

2 3

4 5 6

7 8 9 10

11 12 13 14 15

16 17 18 19 20 21

22 23 24 25 26 27 28

29 30 31 32 33 34 35 36

37 38 39 40 41 42 43 44 45

46 47 48 49 50 51 52 53 54 55

Complete the missing parts so that it creates the output above. Note that the columns of numbers do not need to line up perfectly. Run your program in using python3 to test your work. HINT: Row i has i columns.

def triangle():

value = 1

row = 1

while row <= **\_\_\_\_\_**:

column = 1

while column <= **\_\_\_\_\_\_\_**:

if column != **\_\_\_\_\_\_\_**:

print(value, ' ', sep = '', end = '')

else:

print(value)

value = value + 1

column = column + 1

row = row + 1

**Exercise 1.4**

1. Write the following functions:

* cube(n), which takes in a number and returns its cube. For example, cube(3) => 27.
* factorial(n), which takes in a non-negative integer n and returns n!, which is the product of the integers from 1 to n. (0! = 1 by definition.)
* count\_pattern(pattern lst), which counts the number of times a certain pattern of symbols appears in a list, including overlaps. So count\_pattern( ('a', 'b'), ('a','b', 'c', 'e', 'b', 'a', 'b', 'f')) should return 2, and count\_pattern(('a', 'b', 'a'), ('g', 'a', 'b', 'a', 'b', 'a','b', 'a')) should return 3.
* Write a python program to print the multiplication table for the given number?
* Write a python program to implement Simple Calculator program? (+, -, / ,\*)
* Write a python program to sort the sentence in alphabetical order?

1. Write a Python class to convert an integer to a roman numeral.
2. Write a Python class to find a pair of elements (indices of the two numbers) from a given array whose sum equals a specific target number. Input: numbers= [10,20,10,40,50,60,70], target=50 Output: 3, 4
3. Write a Python class to find the three elements that sum to zero from a set of n real numbers. Input array : [-25, -10, -7, -3, 2, 4, 8, 10] Output : [[-10, 2, 8], [-7, -3, 10]].
4. Write a Python class to reverse a string word by word. Input string : 'hello .py' Expected Output : '.py hello'
5. Count the numbers of characters in the string
   1. Read the string.
   2. Count the characters
   3. Display the result
6. Addition of two square matrices.
   1. Create a lists to read matrix elements
   2. Read the elements of to matrices add the elements
   3. Store the result in third matrix.
   4. Repeat steps 2 and 3 till the addition of all elements
7. Display the result Multiplication of two matrices
   1. Create a lists to read matrix elements
   2. Read the elements of two matrices, multiply the elements
   3. Store the result in third matrix.
   4. Repeat steps 2 and 3 till the multiplication of all elements
   5. Display the result.
8. Write a function called calculator. It should take the following parameters: two numbers, an arithmetic operation (which can be addition, subtraction, multiplication or division and is addition by default), and an output format (which can be integer or floating point, and is floating point by default). Division should be floating-point division. The function should perform the requested operation on the two input numbers, and return a result in the requested format (if the format is integer, the result should be rounded and not just truncated). Raise exceptions as appropriate if any of the parameters passed to the function are invalid.
9. Create a class called Numbers, which has a single class attribute called MULTIPLIER, and a constructor which takes the parameters x and y (these should all be numbers).
   1. Write a method called add which returns the sum of the attributes x and y.
   2. Write a class method called multiply, which takes a single number parameter a and returns the product of a and MULTIPLIER.
   3. Write a static method called subtract, which takes two number parameters, b and c, and returns b - c.
   4. Write a method called value which returns a tuple containing the values of x and y. Make this method into a property, and write a setter and a deleter for manipulating the values of x and y.