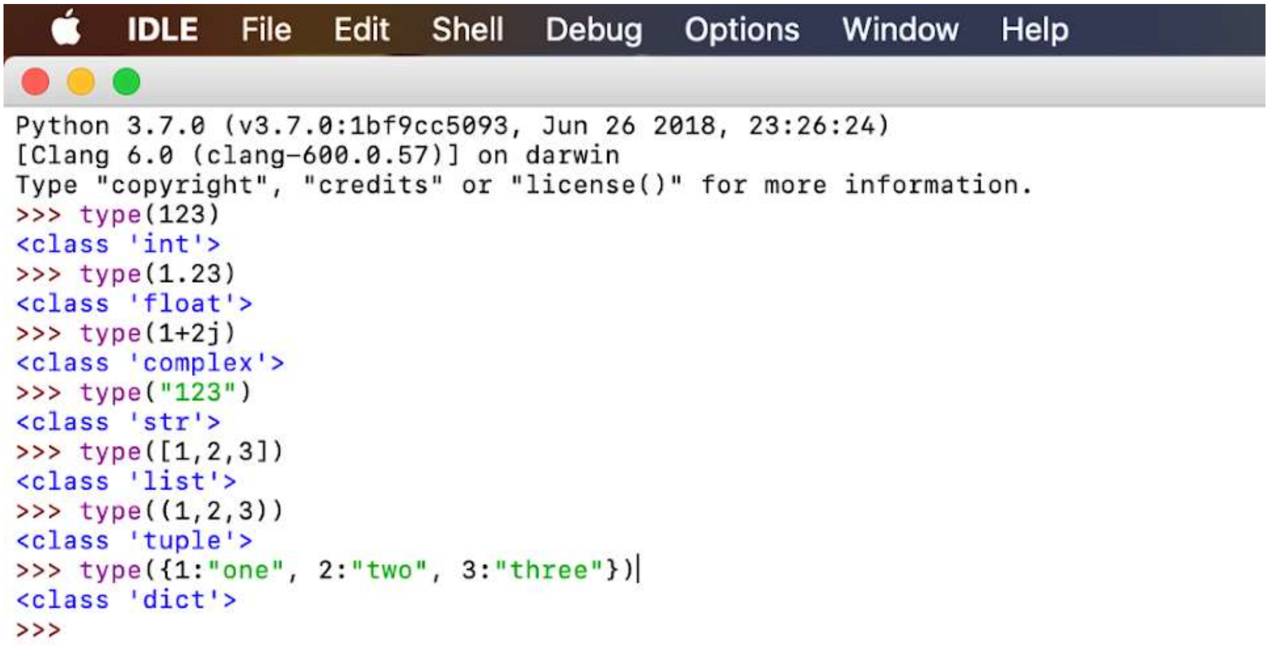
Kaleb Burdin
04/10/2020
Programming Languages

Introduction

There are thousands of spoken languages in the world, each one takes a little bit from another but in whole, they're different in their own unique ways. This can be said the same for the thousands of programming languages developed throughout the years. Each language has its own specific design that enables it to stand out from its competitors but it's not easy implementing whole new designs so each language piggybacks off a few other languages as reference. Two very popular languages today are Python designed by Guido von Russom and C++ designed by Bjarne Stroustrup. Both of these languages are high-level languages that allow you to focus on the functionality of applications.

Comparison: Python vs. C++

Python has the following standard or built-in data types: Numeric, Boolean, Sequence Type, and Dictionary. C++ contains many of the same data types, that being Numeric, Boolean, and Sequence Type. The different Numeric data types of Python are **integers**- positive or negative whole numbers-, **floats**- any real number with a floating-point representation in which a fractional component is denoted by a decimal symbol or scientific notation-, and **complex numbers**- a number with a real and imaginary component represented as x + yj. x and y are floats and j is -1(square root of -1 called an imaginary number). C++ and Python share the same Numeric data types, as do most programming languages due to the need of using such data types. The Boolean data type consists of two built-in values: True or False. The 'T' and 'F' have to be capitalized as if 'true' or 'false' is returned to a Boolean type variable then an error will be thrown. C++ contains the same Boolean types, but the 'T' and 'F' do not have to be capitalized. Thus, 'true' and 'false' are valid and will not throw an error if passed to a Boolean type. The Sequence data types of Python are **strings**- a collection of one or more characters put in single, double or triple quotes-, **lists**- an ordered collection of one or more data items, not necessarily of the same type, put in square brackets-, and **tuples**- an ordered collection of one or more data items, not necessarily of the same type, put in parentheses. This is where C++ seems to differ. Instead of *Lists* and *Tuples*, C++ offers the programmer *Arrays* and *Vectors*. *Lists* and *Arrays* are equivalent while *Vectors* and *Tuples* are equivalent. The Dictionary data type of Python is a unique data type that C++ does not offer. If is an unordered collection of data in a key:value pair form. The popularly used scalar types in Python are: int, float, complex, bool, str, bytes, and NoneType (None). C++ offers the same scalar types except NoneType (None). Instead of this, C++ offers NULL which is equivalent to the NoneType (None) in Python. This makes the remaining data types of Python, *Lists* and *Dictionaries*, some built-in non-scalar types in the language while the remaining data types of C++, *Arrays* and *Vectors*, are some built-in non-scalar types of the language. The big difference between Python and C++ when it comes to variable types, though, is how variable names are bound to the types.

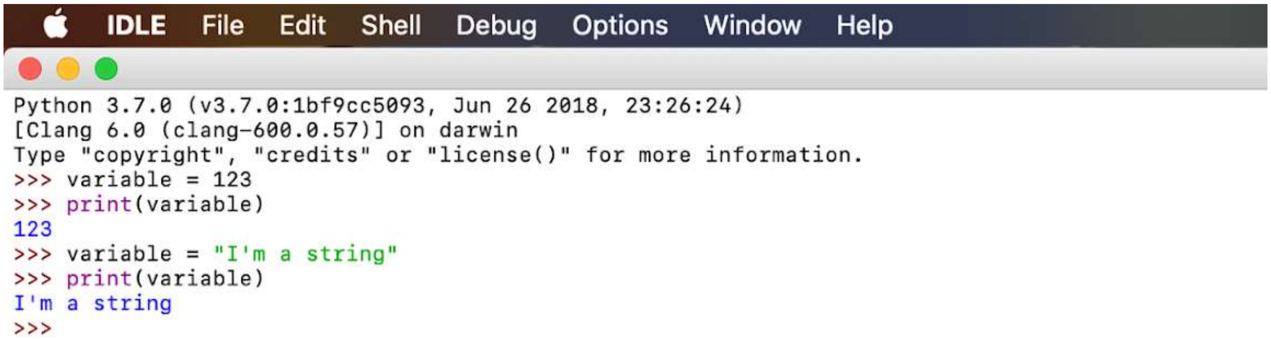


```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> type(123)
<class 'int'>
>>> type(1.23)
<class 'float'>
>>> type(1+2j)
<class 'complex'>
>>> type("123")
<class 'str'>
>>> type([1,2,3])
<class 'list'>
>>> type((1,2,3))
<class 'tuple'>
>>> type({1:"one", 2:"two", 3:"three"})
<class 'dict'>
>>>
```

Here's an example of some of the data types in Python

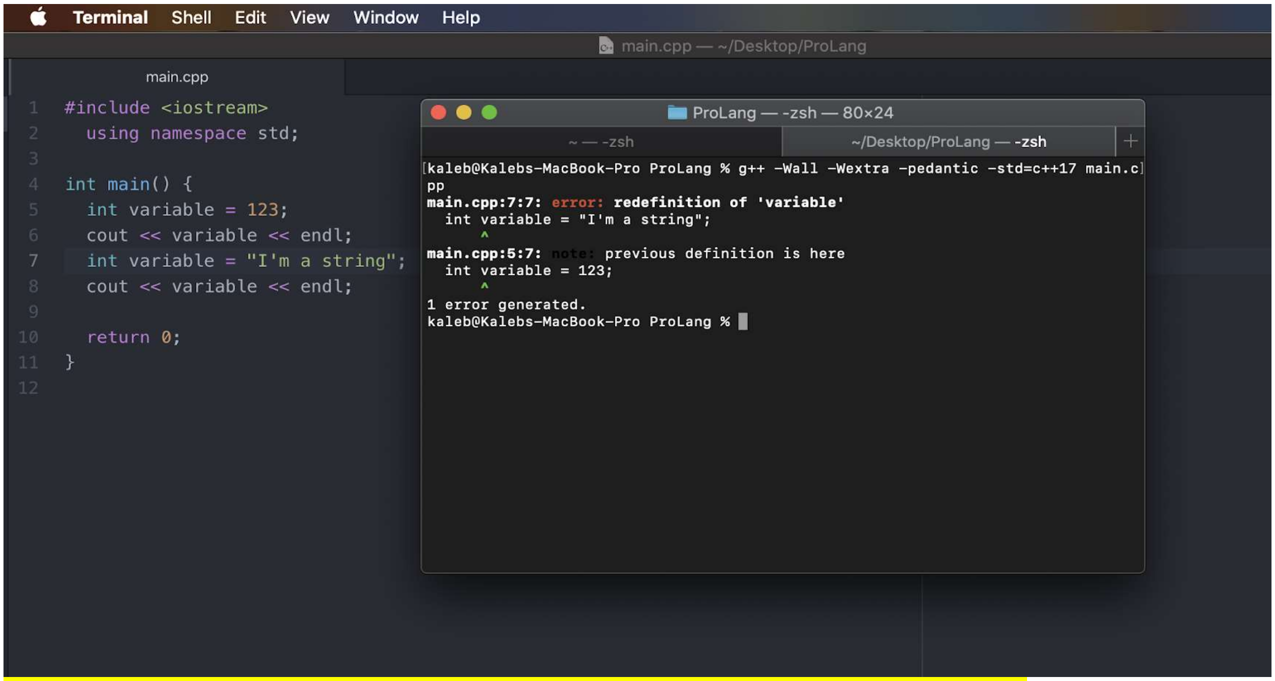
Variable names are loosely bound in Python, meaning that they aren't statically typed; a variable may be assigned a value of one type and then later re-assigned a value of a different type. For example, if a variable name is assigned to an integer (i.e. variable = 123), the same variable name can be assigned to a string (i.e. variable = "I'm a string") at any time. This is much different from C++ as the variables in C++ are statically typed, meaning the variable is initially declared to have a specific data type, and any value assigned to said variable during its lifetime must always have that type. For example, if a variable name is initialized as an *int variable* then said variable name can only be assigned to integers (i.e. variable = 123) and no other data types, such as strings (i.e. variable = "I'm a string"). Both Python and C++ can be considered strongly typed languages. Python is considered strongly typed because the interpreter keeps track of all of the variable types and the language is very dynamic as it rarely uses what it knows to limit variable usage. In Python, you can't perform operations inappropriate to the type of the object - attempting to add numbers to strings will fail. For example, trying to execute 1 + 2 is acceptable and will equal 3 but trying to execute "Hello" + 1 will fail because there is no defined way to add a string and an integer. C++ is also considered strongly typed as two mismatch types are incompatible, such as a string and an integer. Like Python, attempting to add numbers to strings will fail. Thus, trying to execute 1 + 2 is acceptable and will equal 3 but trying to execute "Hello" + 1 will fail because there is no defined way to add a string and an integer. In Python and C++, these variable names are similarly used when it comes to scoping.

Kaleb Burdin
04/10/2020
Programming Languages



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> variable = 123
>>> print(variable)
123
>>> variable = "I'm a string"
>>> print(variable)
I'm a string
>>>
```

Variable names in Python can swiftly change from one type to another (i.e. an integer to a string)

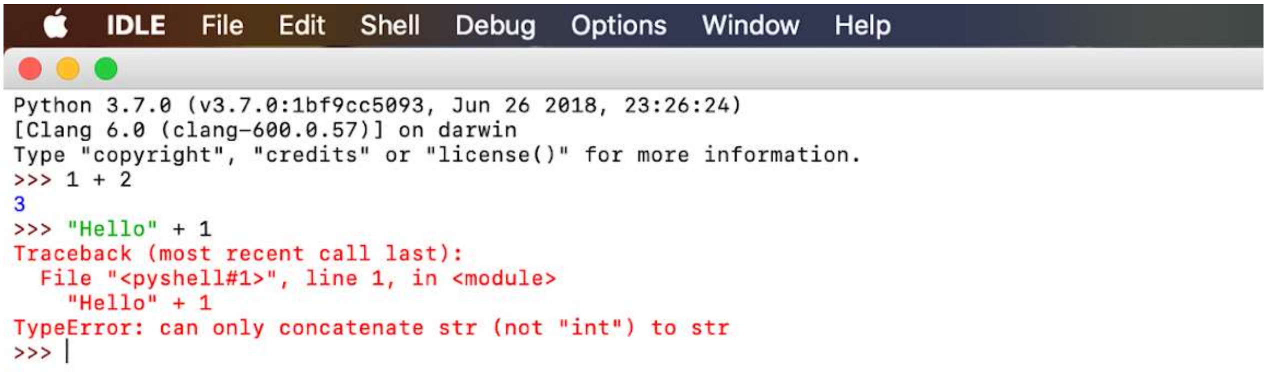


```cpp
#include <iostream>
  using namespace std;

int main() {
  int variable = 123;
  cout << variable << endl;
  int variable = "I'm a string";
  cout << variable << endl;

  return 0;
}
```

```
[kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c]
pp
main.cpp:7:7: error: redefinition of 'variable'
  int variable = "I'm a string";
      ^
main.cpp:5:7: note: previous definition is here
  int variable = 123;
      ^
1 error generated.
kaleb@Kalebs-MacBook-Pro ProLang %
```

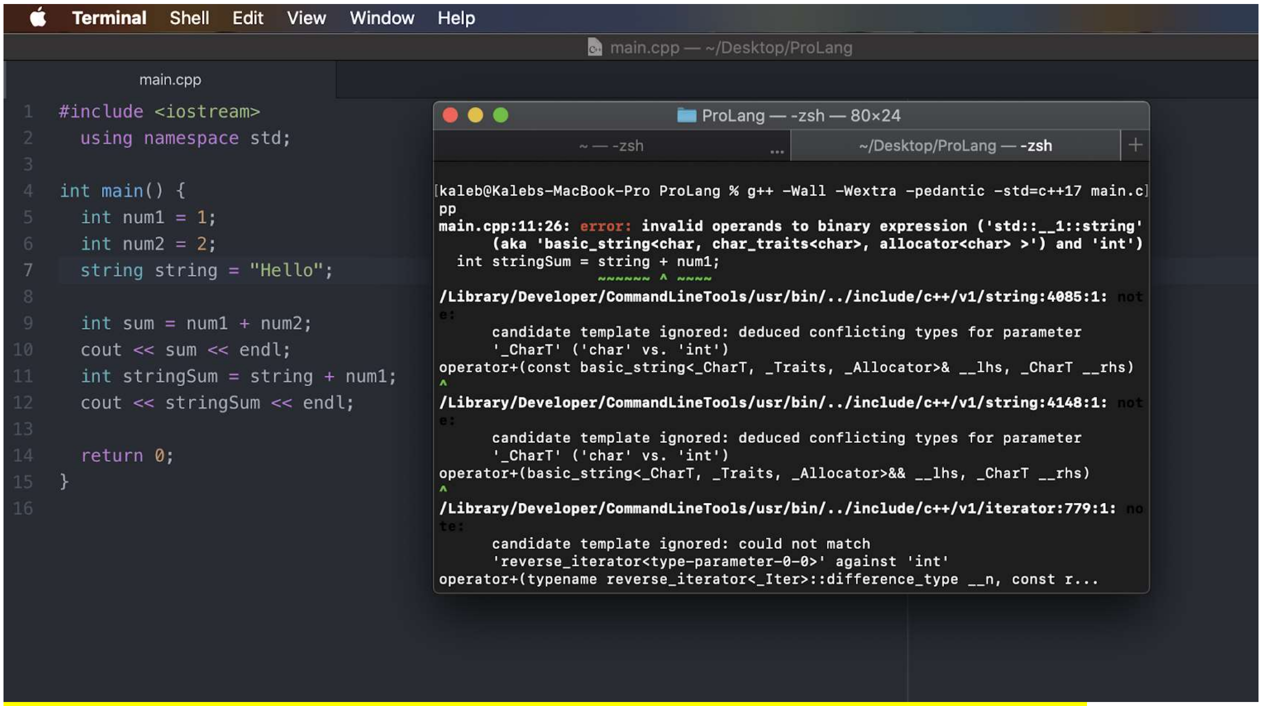
Variable names in C++ are unable to swiftly change from one type to another

Kaleb Burdin
04/10/2020
Programming Languages



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> 1 + 2
3
>>> "Hello" + 1
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    "Hello" + 1
TypeError: can only concatenate str (not "int") to str
>>> |
```

In Python, two integers can be added while a string and an integer cannot

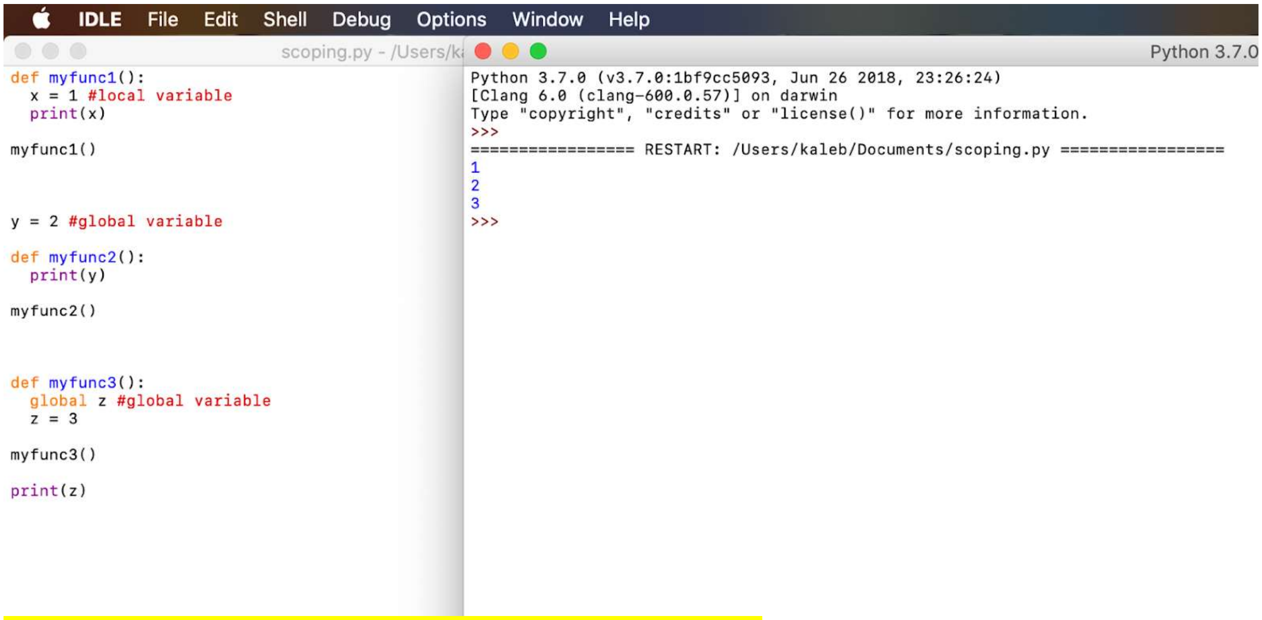


```cpp
#include <iostream>
  using namespace std;

int main() {
  int num1 = 1;
  int num2 = 2;
  string string = "Hello";

  int sum = num1 + num2;
  cout << sum << endl;
  int stringSum = string + num1;
  cout << stringSum << endl;

  return 0;
}
```

```
[kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c]
pp
main.cpp:11:26: error: invalid operands to binary expression ('std::__1::string'
      (aka 'basic_string<char, char_traits<char>, allocator<char> >') and 'int')
  int stringSum = string + num1;
                  ~~~~~~ ^ ~~~~
/Library/Developer/CommandLineTools/usr/bin/../include/c++/v1/string:4085:1: not
e:
      candidate template ignored: deduced conflicting types for parameter
      '_CharT' ('char' vs. 'int')
operator+(const basic_string<_CharT, _Traits, _Allocator>& __lhs, _CharT __rhs)
^
/Library/Developer/CommandLineTools/usr/bin/../include/c++/v1/string:4148:1: not
e:
      candidate template ignored: deduced conflicting types for parameter
      '_CharT' ('char' vs. 'int')
operator+(basic_string<_CharT, _Traits, _Allocator>&& __lhs, _CharT __rhs)
^
/Library/Developer/CommandLineTools/usr/bin/../include/c++/v1/iterator:779:1: no
te:
      candidate template ignored: could not match
      'reverse_iterator<type-parameter-0-0>' against 'int'
operator+(typename reverse_iterator<_Iter>::difference_type __n, const r...
```

A runtime error occurs when attempting to add a string an integer together in C++

A variable is only available from inside the region it is created; this is called the scope of a variable. Both Python and C++ contain both local and global scopes where a variable created inside a function belongs to the *local scope* of that function and can only be used inside that function and a variable created in the main body of the code is a global variable and belongs to the *global scope*. The implementation of a local scope is identical in both languages where if you have a function named myFunction then you can assign a variable called x to the value of 1. By being in the function body, the variable is local to the function and can be used anywhere inside of the function. Global scope is a little different when it comes to the two languages. A similar implementation of a global variable is to assign a variable called x to the value of 1 outside of all functions in the code. By being outside of all of the functions, then every function is able to use and manipulate the variable. Python also implements the keyword *global*. The *global* keyword enables the programmer to initialize and assign a global variable inside of a function to be used in the global scope. Both Python and C++ offer many different function types to implement these variables. Python offers three types of functions: built-in functions- this include print() which prints an object to the terminal-, user-defined functions- helper functions created by the programmer-, and anonymous functions- also called lambda functions because they are not declared with the standard *def* keyword. C++ also offers programmers the ability to use both built-in functions- this includes pow() which computes the power of a number-, and user-defined functions. These functions can be implemented and manipulated by the different classes and objects offered by both Python and C++.

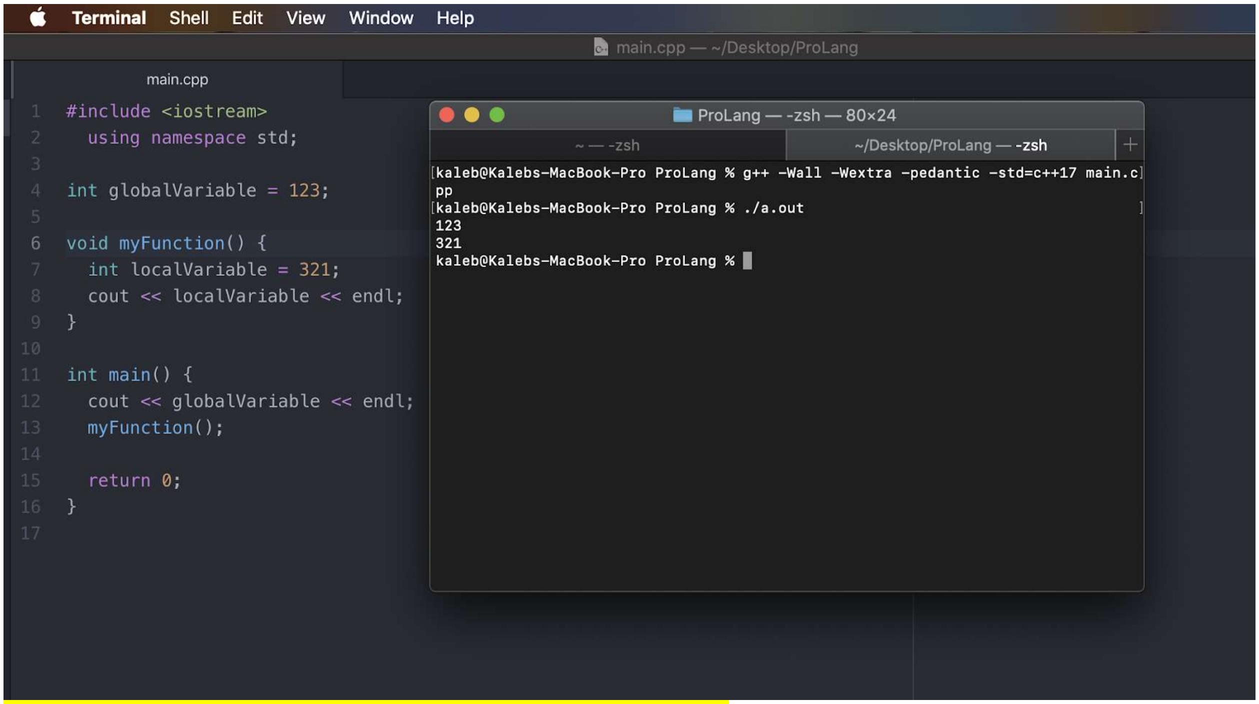


```
def myfunc1():
  x = 1 #local variable
  print(x)

myfunc1()



y = 2 #global variable

def myfunc2():
  print(y)

myfunc2()



def myfunc3():
  global z #global variable
  z = 3

myfunc3()

print(z)
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: /Users/kaleb/Documents/scoping.py =================
1
2
3
>>>
```

Here's an example of local and global scoping in Python

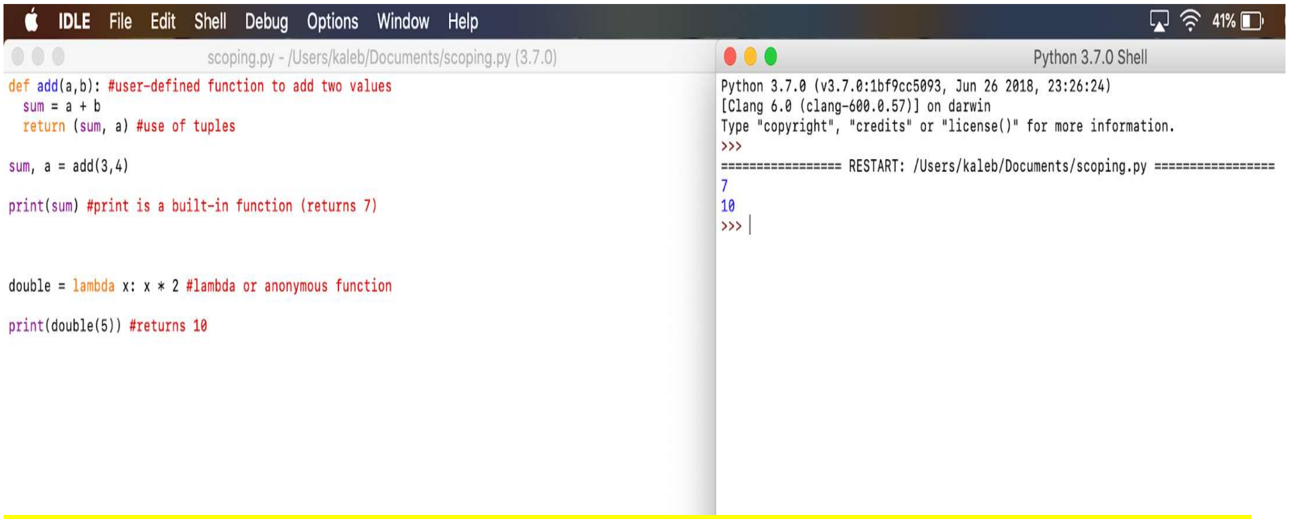
Kaleb Burdin
04/10/2020
Programming Languages



```
#include <iostream>
  using namespace std;

int globalVariable = 123;

void myFunction() {
  int localVariable = 321;
  cout << localVariable << endl;
}

int main() {
  cout << globalVariable << endl;
  myFunction();

  return 0;
}
```

```
[kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c]
pp
[kaleb@Kalebs-MacBook-Pro ProLang % ./a.out                                       ]
123
321
kaleb@Kalebs-MacBook-Pro ProLang %
```

Here’s an example of local and global scoping in C++

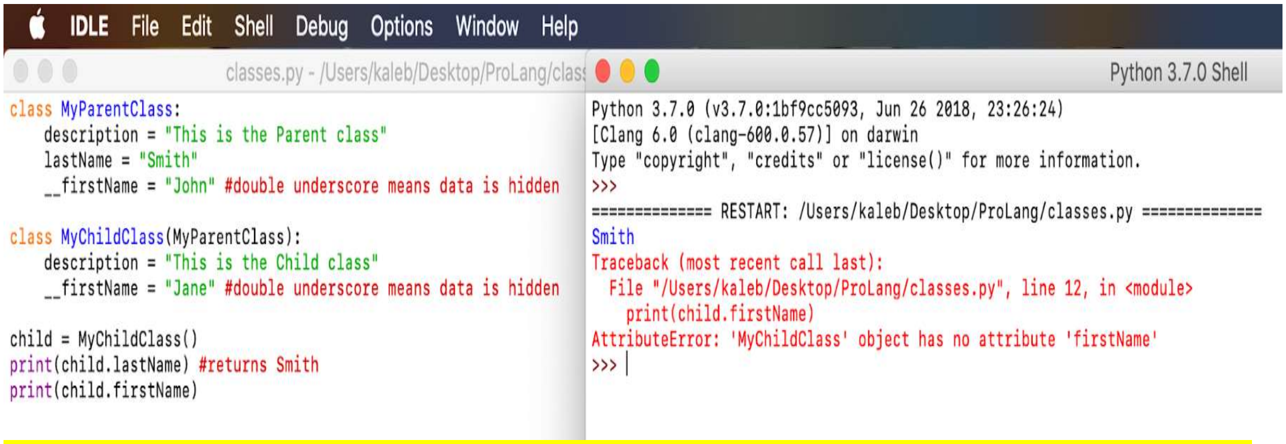


```
def add(a,b): #user-defined function to add two values
  sum = a + b
  return (sum, a) #use of tuples

sum, a = add(3,4)

print(sum) #print is a built-in function (returns 7)



double = lambda x: x * 2 #lambda or anonymous function

print(double(5)) #returns 10
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: /Users/kaleb/Documents/scoping.py ================
7
10
>>> |
```

*User-defined functions are implemented using the *def* keyword while anonymous functions use the *lambda* keyword*

As object-oriented languages, Python and C++ offer both **classes** and **objects** to be used by the programmer. The ***class*** statement is used by both languages; it creates a new class definition in which the name of the class immediately follows the keyword class. In Python, the name is immediately followed by a colon (i.e. class Parent:) while in C++ it’s followed by a curly brace (i.e. class Parent{}). Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute: **_dict_**- a dictionary containing the class's namespace-, **_doc_**- the class documentation string-, **_name_**- the class name-, **_module_**- the module name in which the class is defined-, and **_bases_**- a tuple containing the base classes, in the order of their occurrence in the base class list. Instead of using the normal statements to access attributesPython allows programmers to use the following functions: **getattr(obj,name[,default])**- enables access the attribute of object-, **hasattr(obj,name)**- checks if an attribute exists or not-, **setattr(obj,name,value)**- sets an attribute of an object-, and **delattr(obj, name)**- deletes an attribute of an object. To access attributes of an object in C++, the *dot* operator must be used (i.e. Parent.name). One of the key features of Python and C++ being object-oriented languages is their allowance of class inheritance. In Python, derived classes are declared much like their parent class; however, a list of base classes to inherit from is given after the class name (i.e. class Child (parent):). In C++, inheritance is conducted differently; derived classes are declared much like their parent classes as well and a list of base classes to inherit from is given after the class name, separated by a colon (i.e. class Child : Parent{}). As part of the languages, Python and C++ both enable the programmer to conduct method overriding and method base overloading. Python always allows the programmer to override the parent class methods; this can be done by defining a class method, such as myMethod(self), in class Parent and redefining the method in class Child(Parent). This is the same for C++, but some methods can be used in descendent functions using the keyword *virtual*. A huge object-oriented functionality that Python offers is that of *Data Hiding*. This is similar to adding the *Private* attribute to objects in C++. *Data Hiding* allows the programmer to make an object's attributes only visible within the object. This can be done by naming attributes with a double underscore prefix, such as __hiddenCount = 0 in class Counter. The attribute __hiddenCount is only visible within the class named Counter. Just like there are words that have literal meanings in the English languages, there are objects in programming languages that have literal meanings.

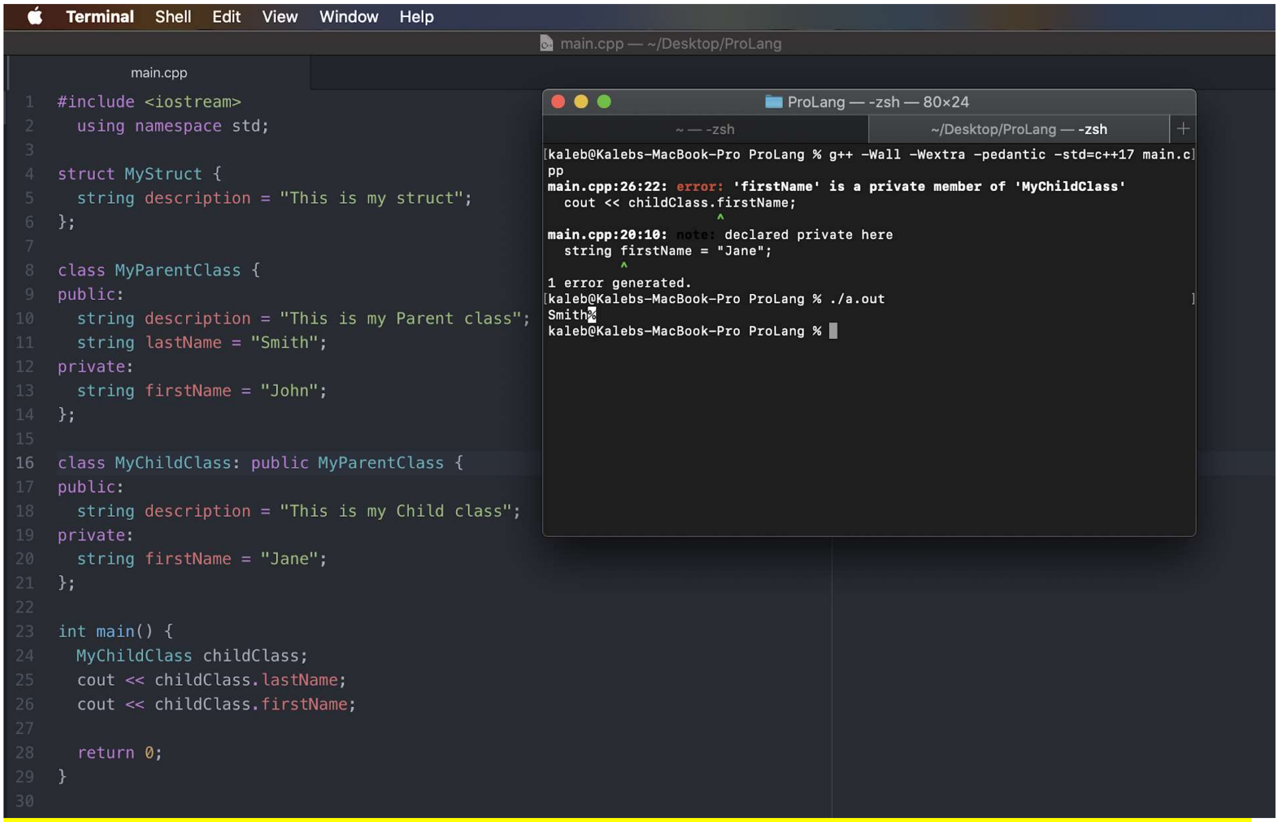


```
class MyParentClass:
    description = "This is the Parent class"
    lastName = "Smith"
    __firstName = "John" #double underscore means data is hidden

class MyChildClass(MyParentClass):
    description = "This is the Child class"
    __firstName = "Jane" #double underscore means data is hidden

child = MyChildClass()
print(child.lastName) #returns Smith
print(child.firstName)
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/classes.py ==============
Smith
Traceback (most recent call last):
  File "/Users/kaleb/Desktop/ProLang/classes.py", line 12, in <module>
    print(child.firstName)
AttributeError: 'MyChildClass' object has no attribute 'firstName'
>>> |
```

Here's an example of the implementation of classes, inheritance, and the data hiding in Python. The Child's first name cannot be retrieved because it's hidden due to the double underscore.

Kaleb Burdin
04/10/2020
Programming Languages

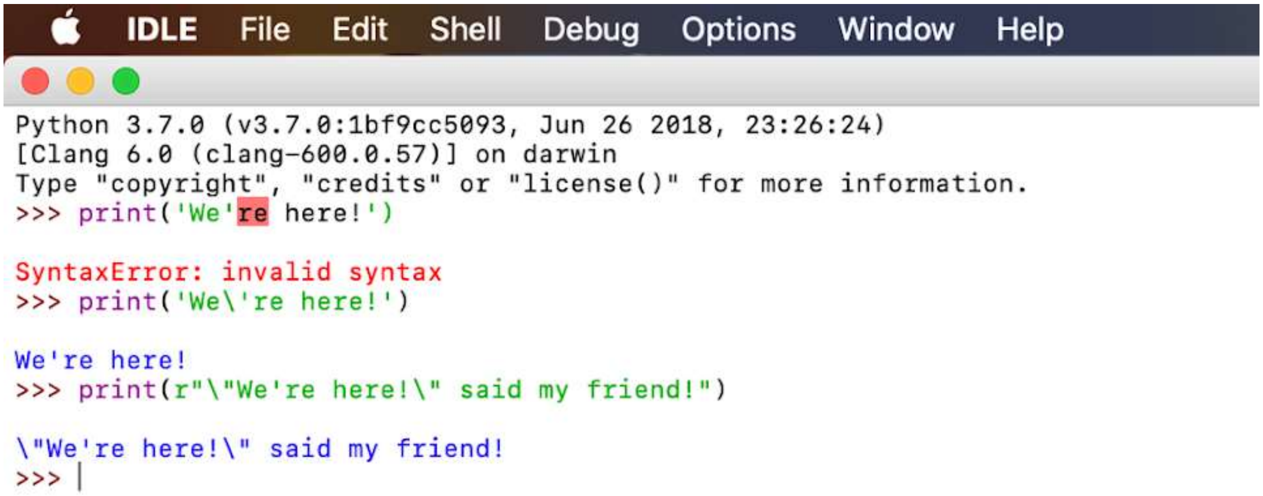


```cpp
#include <iostream>
  using namespace std;

struct MyStruct {
  string description = "This is my struct";
};

class MyParentClass {
public:
  string description = "This is my Parent class";
  string lastName = "Smith";
private:
  string firstName = "John";
};

class MyChildClass: public MyParentClass {
public:
  string description = "This is my Child class";
private:
  string firstName = "Jane";
};

int main() {
  MyChildClass childClass;
  cout << childClass.lastName;
  cout << childClass.firstName;

  return 0;
}
```

```
kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c
pp
main.cpp:26:22: error: 'firstName' is a private member of 'MyChildClass'
  cout << childClass.firstName;
                     ^
main.cpp:20:10: note: declared private here
  string firstName = "Jane";
         ^
1 error generated.
kaleb@Kalebs-MacBook-Pro ProLang % ./a.out
Smith%
kaleb@Kalebs-MacBook-Pro ProLang % 
```

*Here’s an example of the implementation of structs, classes, inheritance, and the *private* keyword in C++. The Child’s first name cannot be retrieved because it’s private.*

In Python, there are five different literals: *string literals*, *numeric literals*, *list literals*, *tuple literals*, and *dictionary literals*. Many of the string literals in Python are similar to the string literals in C++. Both, have, single quotes, double quotes, and triple quotes. These are represented as ‘_’, “_”, and ‘’’_‘’’, respectively. Both Python and C++ can use such examples as literals: ‘Who said that?’, “Who said that?”, and ‘’’Who said that?’’’. Python, though, introduces what’s called *raw strings* and *escaped quotes*. Raw strings are used mostly for regular expressions and are denoted by an r, such as: r “Who said that?”. Escaped quotes are expressed with the backslash character (\), aka the escape character. Prefixing a special character with "\" turns it into an ordinary character. This is called "escaping". For example, "\'" is the single quote character. ‘We’re here!’ therefore is a valid string and equivalent to “We’re here!”. C++ does not have these types of literals, but it does use the backslash character in the same sense that Python does (i.e. /n for a newline). Both Python and C++ specify numeric literals in similar fashions. These numeric literals are integers (i.e. 123), floating-point numbers (i.e. 1.23), negative floating-point numbers (i.e. -1.23), scientific notation (i.e. 1.23E4), hexadecimal notation (i.e. 0x7b), complex numbers (i.e. 1+2*j), and long integers (i.e. 123456789L). The main difference between Python and C++ when it comes to literals is that C++ has literals for *Arrays* while Python has literals for *Lists* and *Dictionaries*. *Lists* in Python act similarly to *Arrays* in Python though as both are specified by assigning a variable name to a set of values enclosed in block braces. C++, though, requires that a value type be initialized with the variable name (i.e. int array = [1, 2, 3]) while Python does not (i.e. array = [1, 2, 3]). *Dictionaries* are a unique object to Python and are specified by a variable name assigned to a set of curly braces in which there are different named strings assigned to different values (i.e. dictionary = {“A”: “Hello”, “B”: “I’m”, “C”: “Kaleb”}).



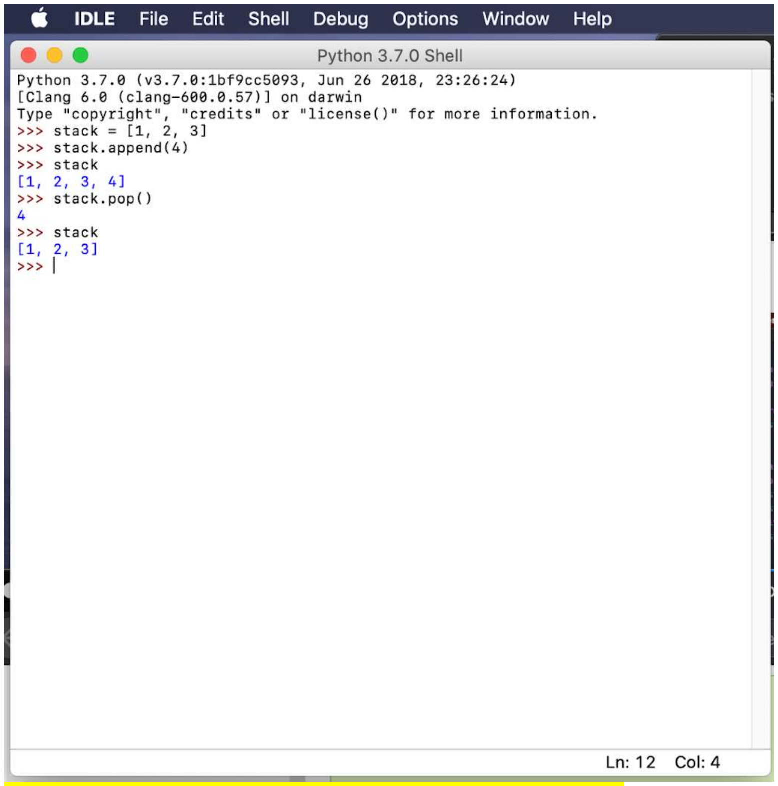
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> print('We're here!')

SyntaxError: invalid syntax
>>> print('We\'re here!')

We're here!
>>> print(r"\"We're here!\" said my friend!")

\"We're here!\" said my friend!
>>> |
```

Here’s an example of a raw string and the escape character in Python

Data Structures and OOP Comparison

A data structure is a particular way of organizing data in a computer so that it can be used effectively. Data structures in C++ are broadly classified into 3 different types: simple data structures, compound data structures, and static and dynamic data structures. Simple data structures are generally built from primitive data types like int, float, double, string, char. For instance, an array is a data structure of similar data type, a structure is also a data structure with the allowance to hold different data types and a class that can hold data elements for various types and member functions as well with any return type. Compound data structures can be built by combining simple data structures. They can be classified into two types: linear data structures and non-linear data structures. A data structure is said to be linear only if it has its elements formed in an ordered sequence. Some of the popular linear data structures that we widely use in C++ are stacks, queues, and linked lists. Non-linear data structures are basically multilevel data structures. Some of the popular non-linear data structures are trees and graphs. Out of these, C++ makes use of arrays, linked lists, stacks, queues, binary trees, binary search trees, heaps, hashing, and many more. *Lists*, *strings* and *tuples* are ordered sequences of objects. Unlike strings that contain only characters, lists and tuples can contain any type of object. *Lists* and *tuples* are like arrays. *Tuples* like strings are immutables. *Lists* are mutable so they can be extended or reduced at will. Sets are mutable unordered sequences of unique elements whereas frozensets are immutable sets. In Python, there are quite a few data structures available. The built-in data structures are: *lists*, *tuples*, *dictionaries*, *strings*, *sets* and *frozensets*. The *lists* can also be used to create *stacks* and *queues*.

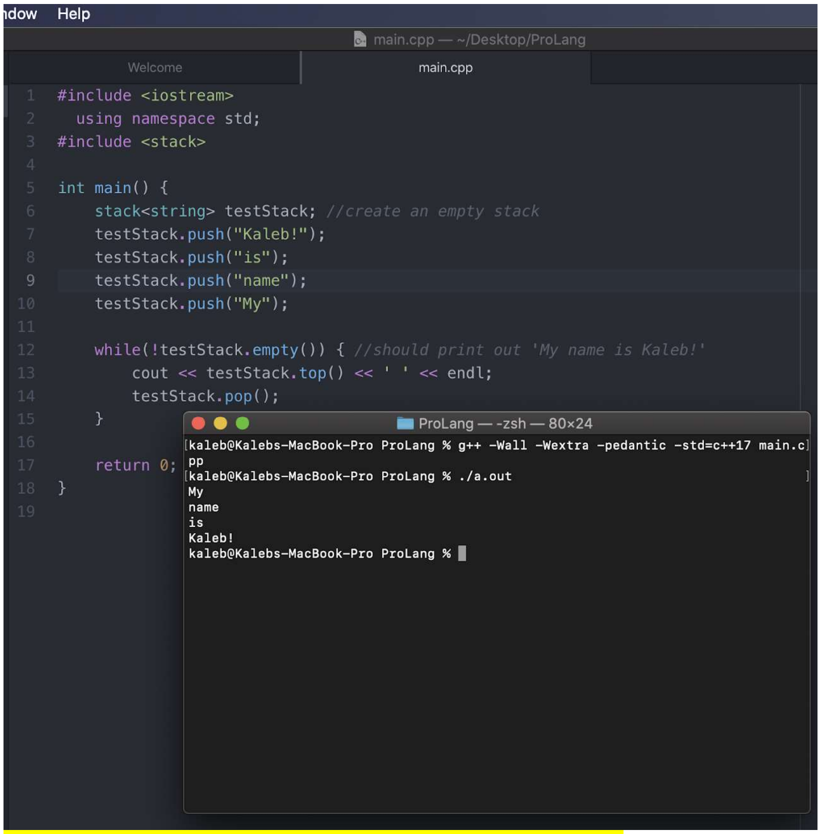
Kaleb Burdin
04/10/2020
Programming Languages



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> stack = [1, 2, 3]
>>> stack.append(4)
>>> stack
[1, 2, 3, 4]
>>> stack.pop()
4
>>> stack
[1, 2, 3]
>>> 
```

Example of the Stack data structure in Python

Kaleb Burdin
04/10/2020
Programming Languages

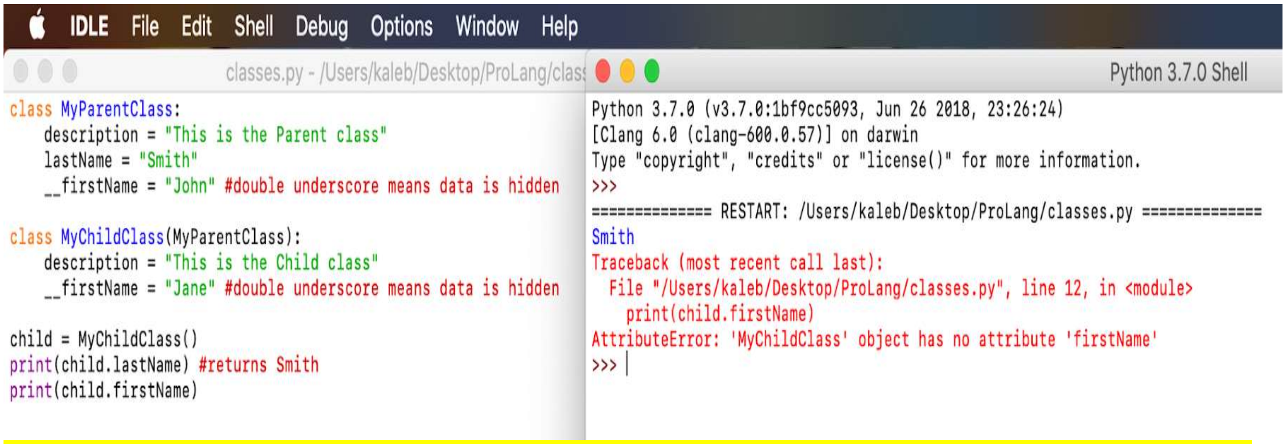


```cpp
#include <iostream>
  using namespace std;
#include <stack>

int main() {
    stack<string> testStack; //create an empty stack
    testStack.push("Kaleb!");
    testStack.push("is");
    testStack.push("name");
    testStack.push("My");

    while(!testStack.empty()) { //should print out 'My name is Kaleb!'
        cout << testStack.top() << ' ' << endl;
        testStack.pop();
    }

    return 0;
}
```

```
[kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c]
pp
[kaleb@Kalebs-MacBook-Pro ProLang % ./a.out                                      ]
My
name
is
Kaleb!
kaleb@Kalebs-MacBook-Pro ProLang % 
```

Example of the Stack data structure in Python

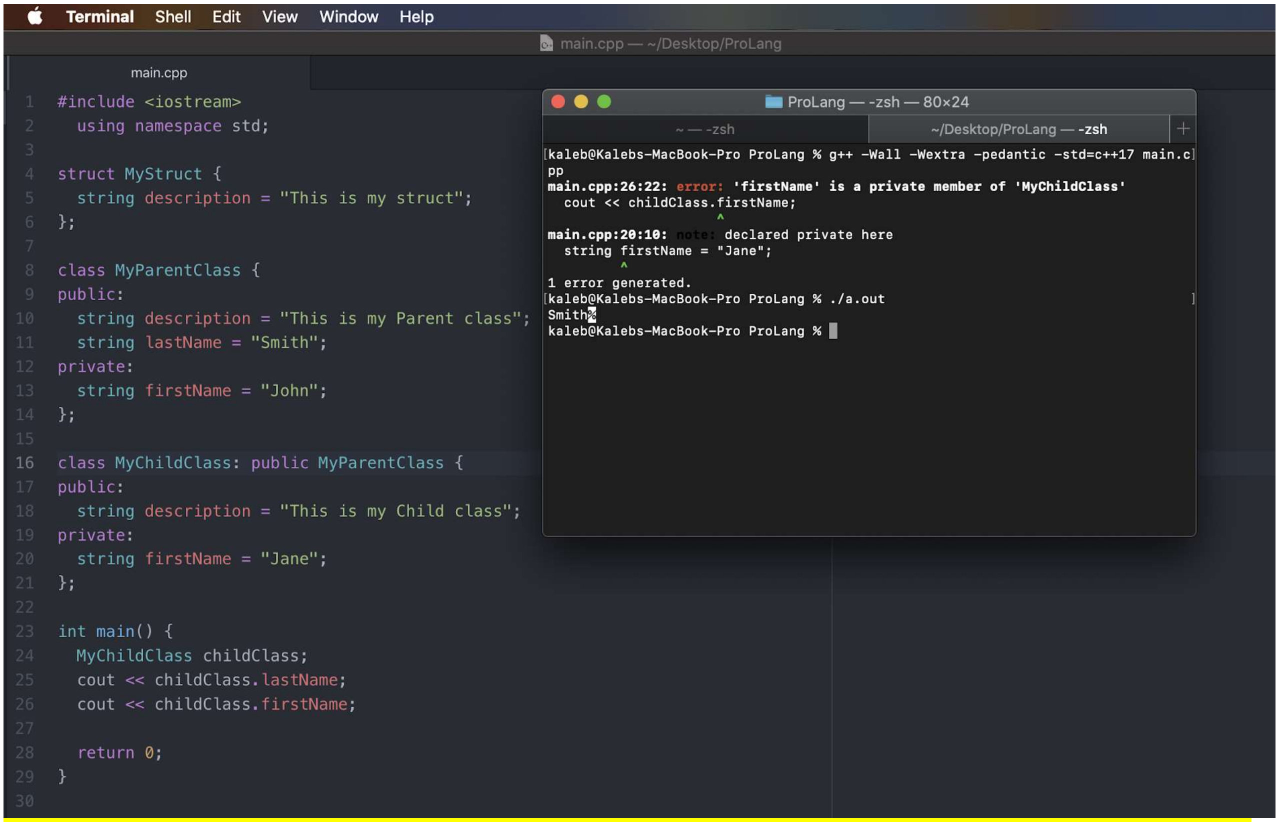
As object-oriented languages, Python and C++ offer both **classes** and **objects** to be used by the programmer. The ***class*** statement is used by both languages; it creates a new class definition in which the name of the class immediately follows the keyword class. In Python, the name is immediately followed by a colon (i.e. class Parent:) while in C++ it's followed by a curly brace (i.e. class Parent{}). Every Python class keeps following built-in attributes and they can be accessed using dot operator like any other attribute: **_dict_**- a dictionary containing the class's namespace-, **_doc_**- the class documentation string-, **_name_**- the class name-, **_module_**- the module name in which the class is defined-, and **_bases_**- a tuple containing the base classes, in the order of their occurrence in the base class list. Instead of using the normal statements to access attributesPython allows programmers to use the following functions: **getattr(obj,name[,default])**-enables access the attribute of object-, **hasattr(obj,name)**- checks if an attribute exists or not-, **setattr(obj,name,value)**- sets an attribute of an object-, and **delattr(obj, name)**- deletes an attribute of an object. To access attributes of an object in C++, the *dot* operator must be used (i.e. Parent.name). One of the key features of Python and C++ being object-oriented languages is their allowance of class inheritance. In Python, derived classes are declared much like their parent class; however, a list of base classes to inherit from is given after the class name (i.e. class Child (parent):). In C++, inheritance is conducted differently; derived classes are declared much like their parent classes as well and a list of base classes to inherit from is given after the class name, separated by a colon (i.e. class Child : Parent{}). As part of the languages, Python and C++ both enable the programmer to conduct method overriding and method base overloading. Python always allows the programmer to override the parent class methods; this can be done by defining a class method, such as myMethod(self), in class Parent and redefining the method in class Child(Parent). This is the same for C++, but some methods can be used in descendent functions using the keyword *virtual*. A huge object-oriented functionality that Python offers is that of *Data Hiding*. This is similar to adding the *Private* attribute to objects in C++. *Data Hiding* allows the programmer to make an object's attributes only visible within the object. This can be done by naming attributes with a double underscore prefix, such as __hiddenCount = 0 in class Counter. The attribute __hiddenCount is only visible within the class named Counter. Just like there are words that have literal meanings in the English languages, there are objects in programming languages that have literal meanings.



```
class MyParentClass:
    description = "This is the Parent class"
    lastName = "Smith"
    __firstName = "John" #double underscore means data is hidden

class MyChildClass(MyParentClass):
    description = "This is the Child class"
    __firstName = "Jane" #double underscore means data is hidden

child = MyChildClass()
print(child.lastName) #returns Smith
print(child.firstName)
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/classes.py ==============
Smith
Traceback (most recent call last):
  File "/Users/kaleb/Desktop/ProLang/classes.py", line 12, in <module>
    print(child.firstName)
AttributeError: 'MyChildClass' object has no attribute 'firstName'
>>> |
```

Here's an example of the implementation of classes, inheritance, and the data hiding in Python. The Child's first name cannot be retrieved because it's hidden due to the double underscore.

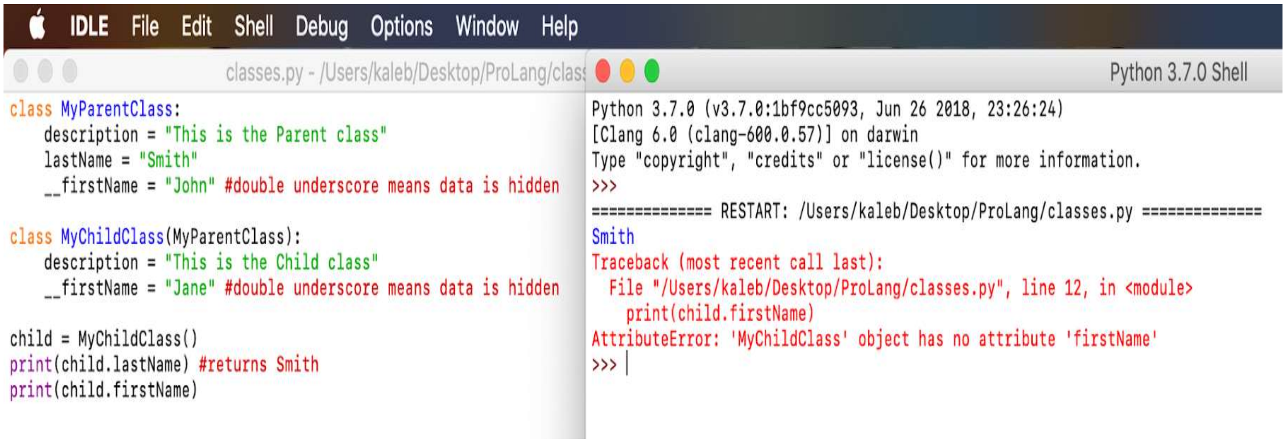


```cpp
#include <iostream>
  using namespace std;

struct MyStruct {
  string description = "This is my struct";
};

class MyParentClass {
public:
  string description = "This is my Parent class";
  string lastName = "Smith";
private:
  string firstName = "John";
};

class MyChildClass: public MyParentClass {
public:
  string description = "This is my Child class";
private:
  string firstName = "Jane";
};

int main() {
  MyChildClass childClass;
  cout << childClass.lastName;
  cout << childClass.firstName;

  return 0;
}
```

```
kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c
pp
main.cpp:26:22: error: 'firstName' is a private member of 'MyChildClass'
  cout << childClass.firstName;
                     ^
main.cpp:20:10: note: declared private here
  string firstName = "Jane";
         ^
1 error generated.
kaleb@Kalebs-MacBook-Pro ProLang % ./a.out
Smith%
kaleb@Kalebs-MacBook-Pro ProLang % 
```

*Here's an example of the implementation of structs, classes, inheritance, and the *private* keyword in C++. The Child's first name cannot be retrieved because it's private.*

A huge object-oriented functionality that Python offers is that of *Data Hiding*. This is similar to adding the *Private* attribute to objects in C++. *Data Hiding* allows the programmer to make an object's attributes only visible within the object. This can be done by naming attributes with a double underscore prefix, such as __hiddenCount = 0 in class Counter. The attribute __hiddenCount is only visible within the class named Counter. Just like there are words that have literal meanings in the English languages, there are objects in programming languages that have literal meanings.

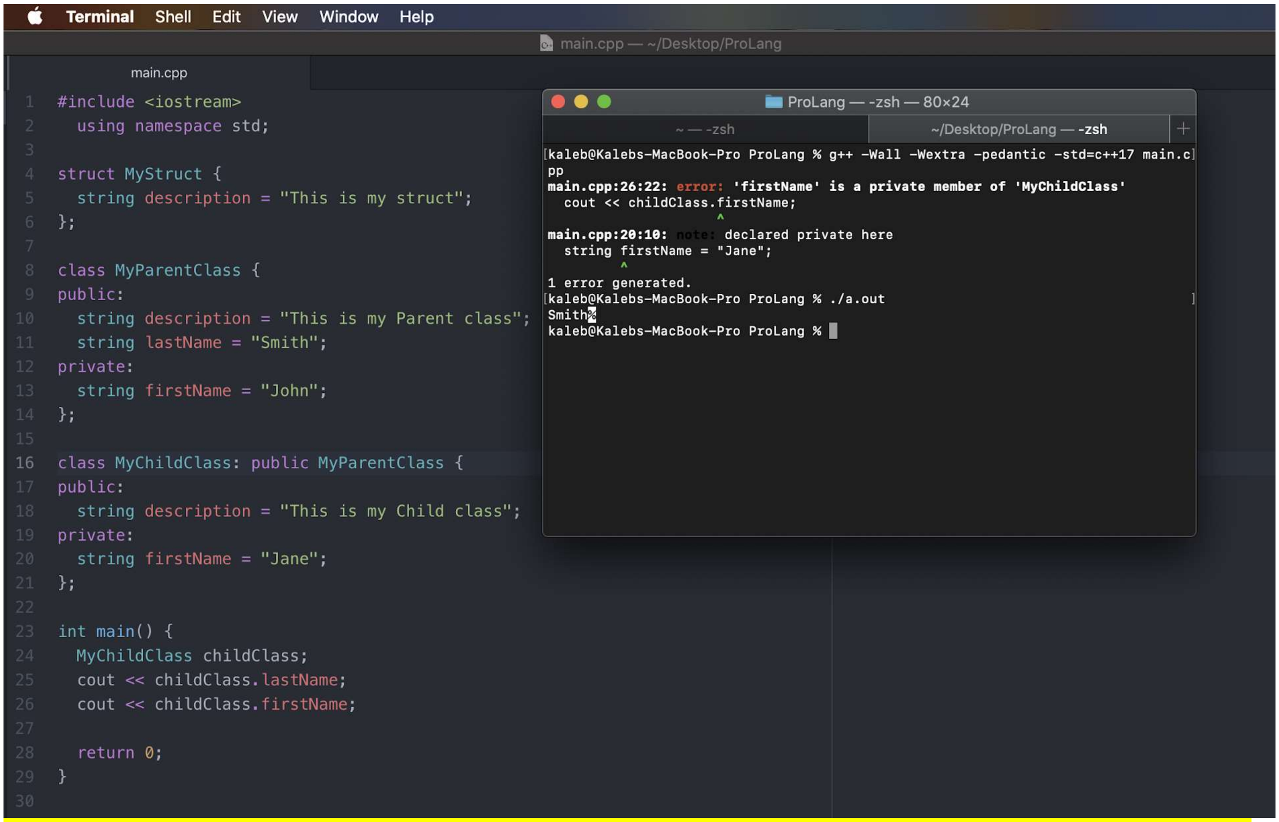


```python
class MyParentClass:
    description = "This is the Parent class"
    lastName = "Smith"
    __firstName = "John" #double underscore means data is hidden

class MyChildClass(MyParentClass):
    description = "This is the Child class"
    __firstName = "Jane" #double underscore means data is hidden

child = MyChildClass()
print(child.lastName) #returns Smith
print(child.firstName)
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/classes.py ==============
Smith
Traceback (most recent call last):
  File "/Users/kaleb/Desktop/ProLang/classes.py", line 12, in <module>
    print(child.firstName)
AttributeError: 'MyChildClass' object has no attribute 'firstName'
>>> 
```

Kaleb Burdin
04/10/2020
Programming Languages

Here's an example of the implementation of classes, inheritance, and the data hiding in Python. The Child's first name cannot be retrieved because it's hidden due to the double underscore.



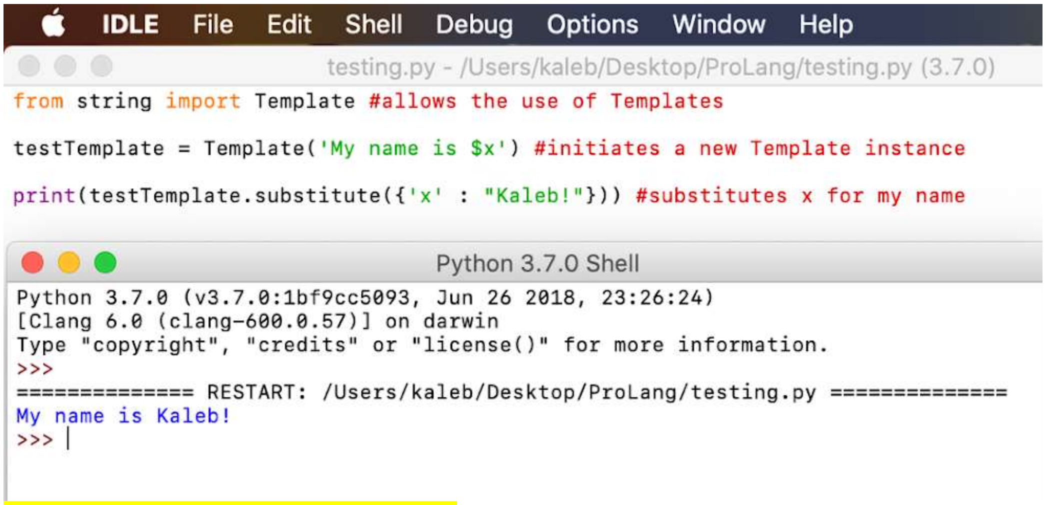
```cpp
#include <iostream>
  using namespace std;

struct MyStruct {
  string description = "This is my struct";
};

class MyParentClass {
public:
  string description = "This is my Parent class";
  string lastName = "Smith";
private:
  string firstName = "John";
};

class MyChildClass: public MyParentClass {
public:
  string description = "This is my Child class";
private:
  string firstName = "Jane";
};

int main() {
  MyChildClass childClass;
  cout << childClass.lastName;
  cout << childClass.firstName;

  return 0;
}
```

```
kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c
pp
main.cpp:26:22: error: 'firstName' is a private member of 'MyChildClass'
  cout << childClass.firstName;
                     ^
main.cpp:20:10: note: declared private here
  string firstName = "Jane";
         ^
1 error generated.
kaleb@Kalebs-MacBook-Pro ProLang % ./a.out
Smith%
kaleb@Kalebs-MacBook-Pro ProLang % 
```

*Here's an example of the implementation of structs, classes, inheritance, and the *private* keyword in C++. The Child's first name cannot be retrieved because it's private.*

Templates are powerful features which allow programmers to write generic programs. In simple terms, a programmer can create a single function or a class to work with different data types using templates. Both C++ and Python support *Templates* but in their own special ways. In C++, there are two types of *Templates*: function templates and class templates. A function template starts with the keyword *template* followed by template parameters inside <> which is followed by function declaration (i.e. template <class Name>). In Python, a template is a class of String module; it allows for data to change without having to edit the application. A template class takes a string as a template, within the string use placeholder variable name preceding the '$' symbol to depict there is the placeholder. Templates are implemented by assigning a variable to Template followed by a string of characters.

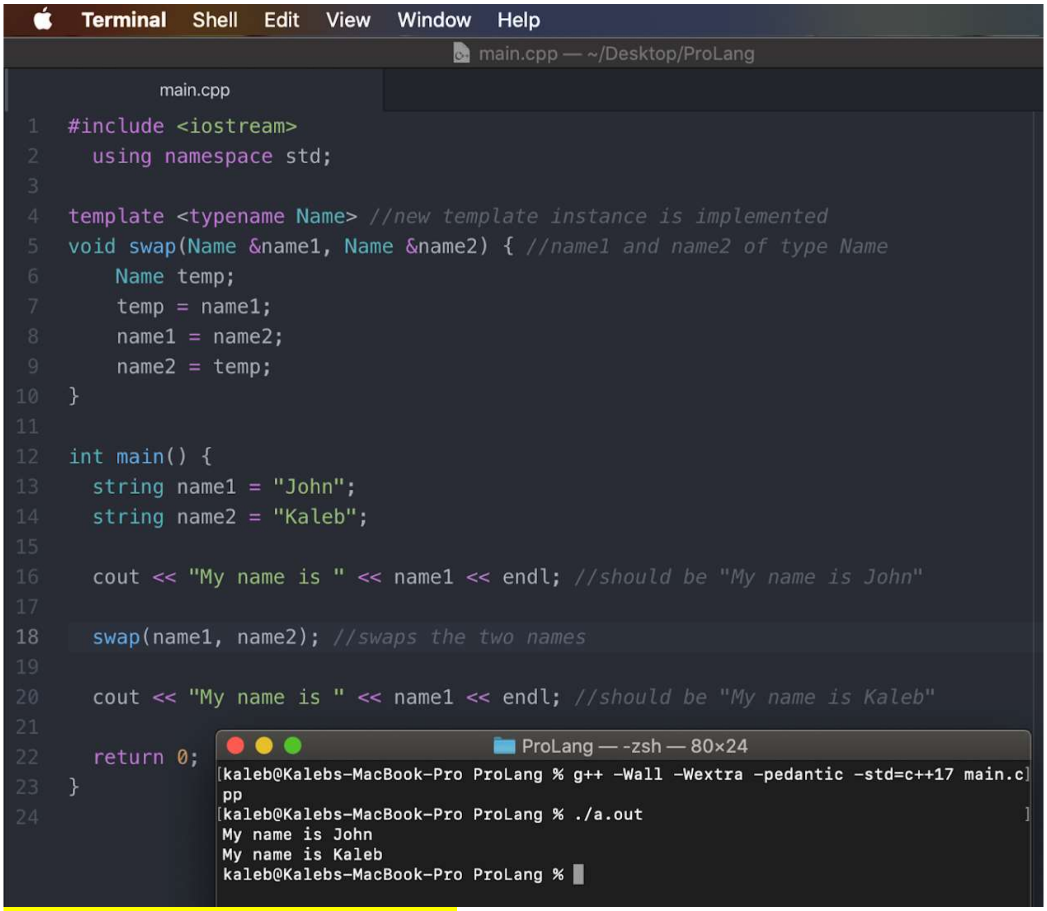
Kaleb Burdin
04/10/2020
Programming Languages


IDLE
File
Edit
Shell
Debug
Options
Window
Help
testing.py - /Users/kaleb/Desktop/ProLang/testing.py (3.7.0)
Python 3.7.0 Shell

```python
from string import Template #allows the use of Templates

testTemplate = Template('My name is $x') #initiates a new Template instance

print(testTemplate.substitute({'x' : "Kaleb!"})) #substitutes x for my name
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/testing.py ==============
My name is Kaleb!
>>> |
```

Example of Templates in Python

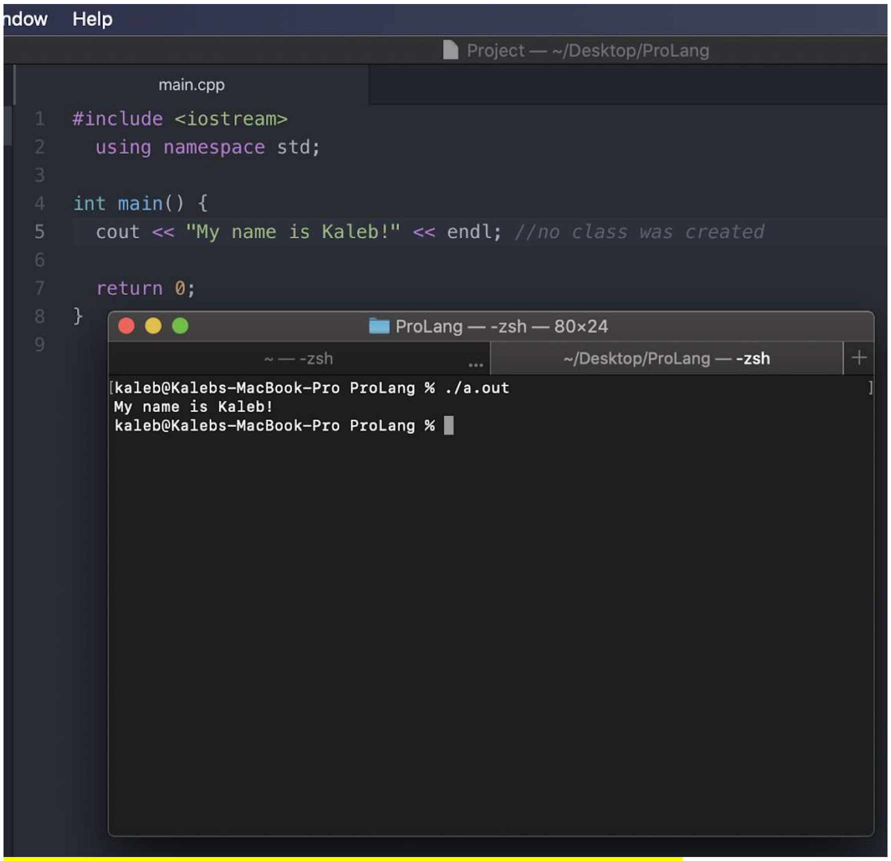


```cpp
#include <iostream>
  using namespace std;

template <typename Name> //new template instance is implemented
void swap(Name &name1, Name &name2) { //name1 and name2 of type Name
    Name temp;
    temp = name1;
    name1 = name2;
    name2 = temp;
}

int main() {
  string name1 = "John";
  string name2 = "Kaleb";

  cout << "My name is " << name1 << endl; //should be "My name is John"

  swap(name1, name2); //swaps the two names

  cout << "My name is " << name1 << endl; //should be "My name is Kaleb"

  return 0;
}
```

```
[kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c]
pp
[kaleb@Kalebs-MacBook-Pro ProLang % ./a.out                                       ]
My name is John
My name is Kaleb
kaleb@Kalebs-MacBook-Pro ProLang % █
```

Example of Templates in Python

The major principles of object-oriented programming systems are as given: objects, classes, methods, inheritance, polymorphism, data abstraction, and encapsulation. A programming language must have strong support for these features in order to be considered purely object-oriented. By that logic, neither C++ nor Python are purely object-oriented programming languages. Python supports object-orientation by providing classes to encapsulate data and functions. You can create objects by instantiating classes. Objects can send messages to each other by means of the method call syntax and the return value mechanism. So, at the basic level, Python is object-oriented. Python supports specialization of classes via inheritance, both single and multiple. Python also supports meta-programming so that you can even control how classes create objects. The kicker, though, is that Python does not enforce strong encapsulation. Python offers limited support for data hiding through naming conventions and through the use of properties, which in combination, can control read/write access. But this is not a commonly used mechanism in the Python community, but it isn't enough to be considered encapsulation and thus Python does not completely support encapsulation, so it is not purely object-oriented. C++ supports object-oriented programming (i.e. inheritance, polymorphism, and encapsulation), but object-orientation is not intrinsic to the language. You can write a valid C++ program without using an object even once. In C++, the main function is mandatory, which executes first, but it resides outside the class and from there we create objects. So, creating a class becomes optional and we can write code without using class. Thus, C++ is not purely object-oriented.



```cpp
#include <iostream>
  using namespace std;

int main() {
  cout << "My name is Kaleb!" << endl; //no class was created

  return 0;
}
```

```
kaleb@Kalebs-MacBook-Pro ProLang % ./a.out
My name is Kaleb!
kaleb@Kalebs-MacBook-Pro ProLang %
```

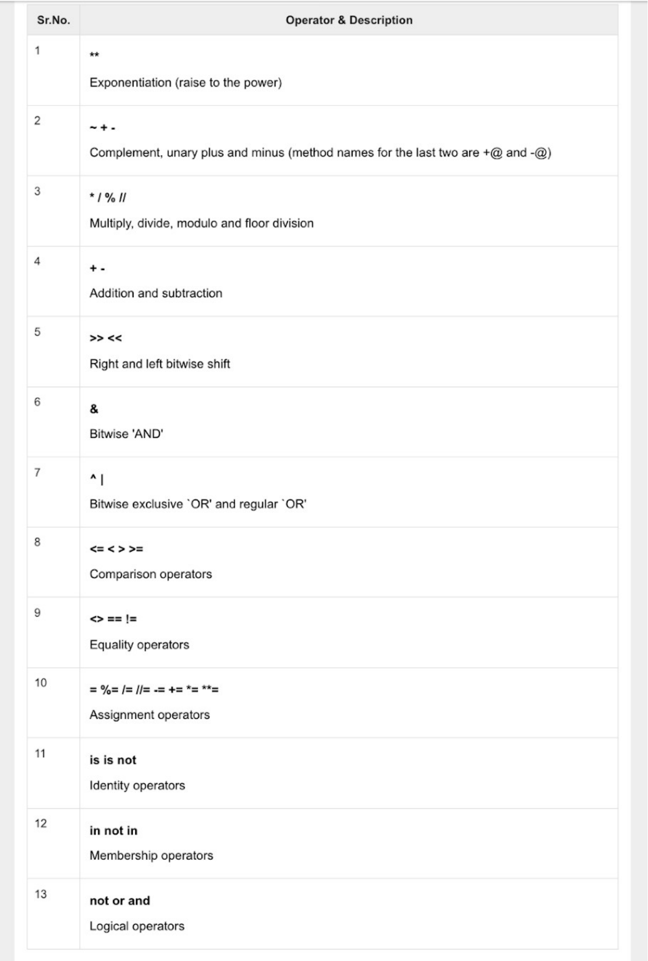
Example of why C++ is not purely object-oriented

Expressions and Statements Comparison

Kaleb Burdin
04/10/2020
Programming Languages

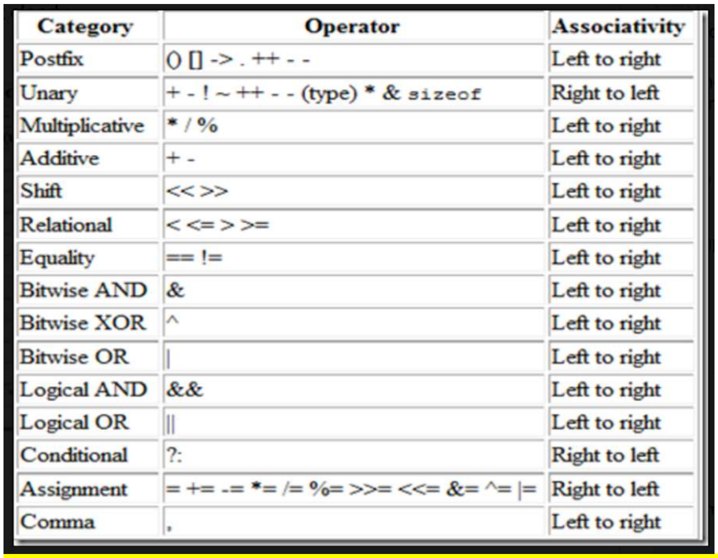
Both Python and C++ provide the programmer with the following types of operators: Arithmetic Operators, Comparison (Relational) Operators, Assignment Operators, Logical Operators, and Bitwise Operators. Python, though, also provides two other types of operators: Membership Operators and Identity Operators. For Python and C++, Arithmetic Operators work with numeric data types (i.e. integers, floats, etc.) and include the following operators: + *Addition* (i.e. 1+1=2), - *Subtraction* (i.e. 1-1=0), * *Multiplication* (i.e. 1*2=2), and % *Modulus* (i.e. 1%2=1). Python and C++ provide their own unique arithmetic operators, though, as Python provides ** *Exponent* (i.e. 2**2=4)- this is equivalent to the pow() function in C++-, and // *Floor Division* (i.e. 9//2=4)- this is equivalent to the floor() function in C++-, while C++ offers ++ *Increment* (i.e. ++x) and – *Decrement* (i.e. --x). For Python and C++, Comparison Operators work with numeric data types (i.e. integers, floats, etc.) **and?** and include the following operators: == *Equal* (i.e. 10==20 returns false), != *Not Equal* (i.e. 10!=20 returns true), > *Greater Than* (i.e. 10>20 returns false), < *Less Than* (i.e. 10<20 returns true), >= *Greater Than Or Equal To* (i.e. 10>=20 returns false), and <= *Less Than Or Equal To* (i.e. 10<=20 returns true). Python offers one unique comparison operator which is <> *Not Equal* (10<>20 returns true)- this is similar to != in both Python and C++. For Python and C++, Assignment Operators work with numeric data types (i.e. integers, floats, etc.) **and?** and include the following operators: = *Assignment* (i.e. c=a+b), += *Add AND* (i.e. c+=a or c=c+a), -= *Subtract AND* (i.e. c-=a or c=c-a), *= *Multiply AND* (i.e. c*=a or c=c*a), /= *Divide AND* (i.e. c/=a or c=c/a), and %= *Modulus AND* (i.e. c%=a or c=c%a). Python and C++ provide their own unique assignment operators, though, as Python provides **= *Exponent AND* (i.e. c**=a or c=c**a) and //= *Floor Division* (i.e. c//=a or c=c//a). For Python and C++, Logical Operators work with numeric data types (i.e. integers, floats, etc.) **and?** and include the following operators for Python: and *Logical AND* (i.e. 10 and 20 return true), or *Logical OR* (i.e. 10 and 20 return true), and not *Logical NOT* (i.e. Not(10 and 20) returns false). C++, instead, has the following operators: && *Logical AND* (i.e. x<5 && x<10), || *Logical OR* (i.e. x<5 || x<4), and ! *Logical NOT* (i.e. !(x<5 && x<10)). For Python and C++, Bitwise Operators include the following operators: & *Binary AND* (i.e. a&b), | *Binary OR* (i.e. a|b), ^ *Binary XOR* (i.e. a^b), ~ *Binary Ones Complement* (i.e. ~a), << *Binary Left Shift* (i.e. a<<2), and >> *Binary Right Shift* (i.e. a>>2). For Python, Membership Operators includes the following operators: *in*- evaluates to true if it finds a variable in the specified sequence and false otherwise-, and *not in*- evaluates to true if it does not find a variable in the specified sequence and false otherwise. For Python, Identity Operators includes the following operators: *is*- evaluates to true if the variables on either side of the operator points to the same object and false otherwise-, and *is not*- evaluates to false if the variables on either side of the operator points to the same object and true otherwise.

Kaleb Burdin
04/10/2020
Programming Languages



Sr.No.	Operator & Description
1	** Exponentiation (raise to the power)
2	~ + - Complement, unary plus and minus (method names for the last two are +@ and -@)
3	* / % // Multiply, divide, modulo and floor division
4	+ - Addition and subtraction
5	>> << Right and left bitwise shift
6	& Bitwise 'AND'
7	^ | Bitwise exclusive `OR' and regular `OR'
8	<= < > >= Comparison operators
9	<> == != Equality operators
10	= %= /= //= -= += *= **= Assignment operators
11	is is not Identity operators
12	in not in Membership operators
13	not or and Logical operators

Precedence chart of Python operators (highest precedence to lowest precedence)

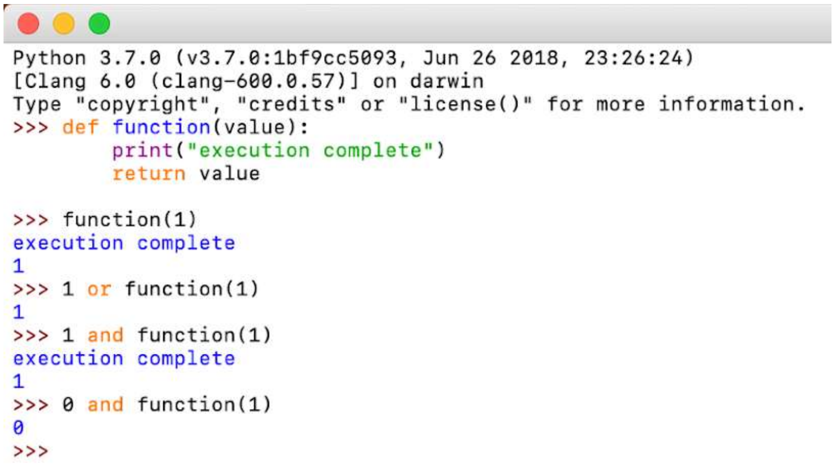


Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Precedence chart of C++ operators (highest precedence to lowest precedence)

Kaleb Burdin
04/10/2020
Programming Languages

Python and C++ both support short-circuit evaluation in which the Python supports it through the operators *and* and *or* and C++ supports it through the operators *&&* and ||. An example of short-circuit evaluation in Python would be creating a function, *def function(value):*, that prints out the message "execution complete" and returns the value *value*. By calling function(1), both the message and the value will be printed. Due to short-circuiting, calling 1 or function(1), only 1 will be printed as 1 evaluates to True so the second value won't be evaluated and while calling 0 and function(1), only 0 will be printed. But, by calling 1 and function(1), both the message and the value will be printed as both evaluate to True. C++ is similar when it comes to examples. Say you have two bools, bool1 and bool2. So, if you have the expression (bool1 && bool2), what happens? If bool1 returns false then bool2 isn't even evaluated but if it returns true then bool2 will be evaluated. If you have the expression (bool1 || bool2), if bool1 returns false then bool2 will still be evaluated but if bool2 returns false then nothing will be returned as both values are false.

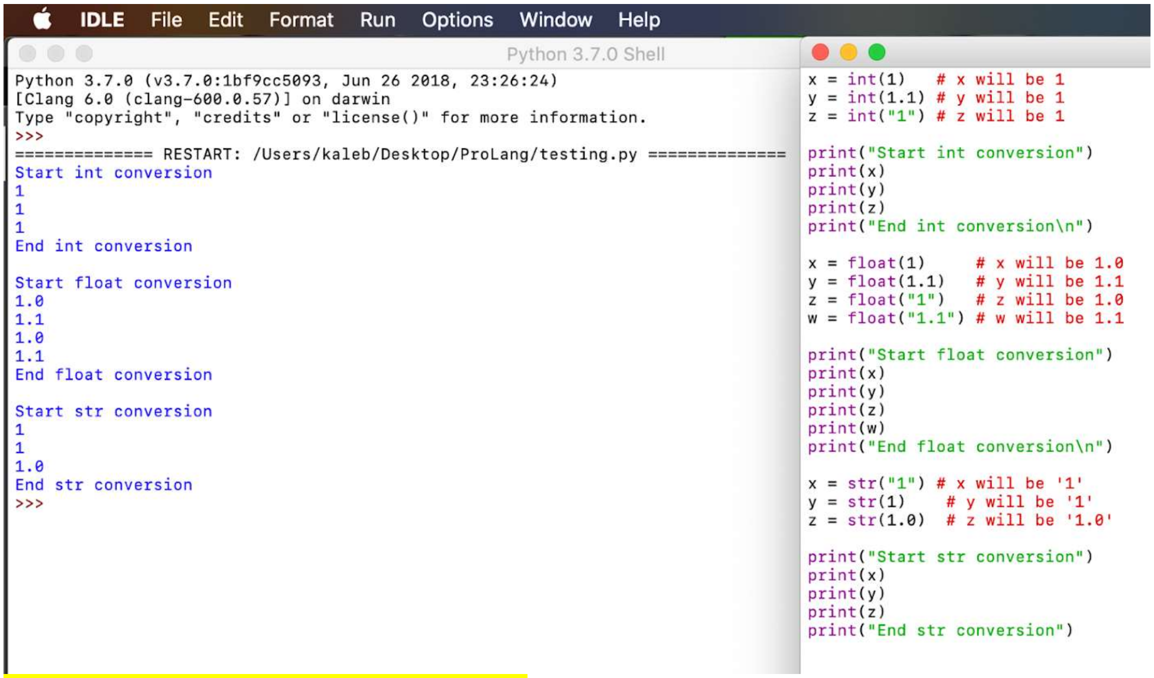


```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>> def function(value):
        print("execution complete")
        return value

>>> function(1)
execution complete
1
>>> 1 or function(1)
1
>>> 1 and function(1)
execution complete
1
>>> 0 and function(1)
0
>>>
```

Example of short-circuiting in Python

A cast is a special operator that forces one data type to be converted into another. As a operator, a cast is unary and has the same precedence as any other unary operator. Python and C++ handling type casting a bit differently as there aren't conversion operators in Python like in C++ because Python does not have a strong static type system. Thus, Python relies on functions such as int(), float(), and str() to execute type conversion. C++, though, does have conversion operators and the most general cast supported by most C++ compilers is *(type) expression*. The list includes the following: const_cast<type>(expr)- used to explicitly override const and/or volatile in a cast-, dynamic_cast<type>(expr)- performs a runtime cast that verifies the validity of the cast-, reinterpret_cast<type>(expr)- changes a pointer to any other type of pointer-, and static_cast<type>(expr)- performs a non polymorphic cast.

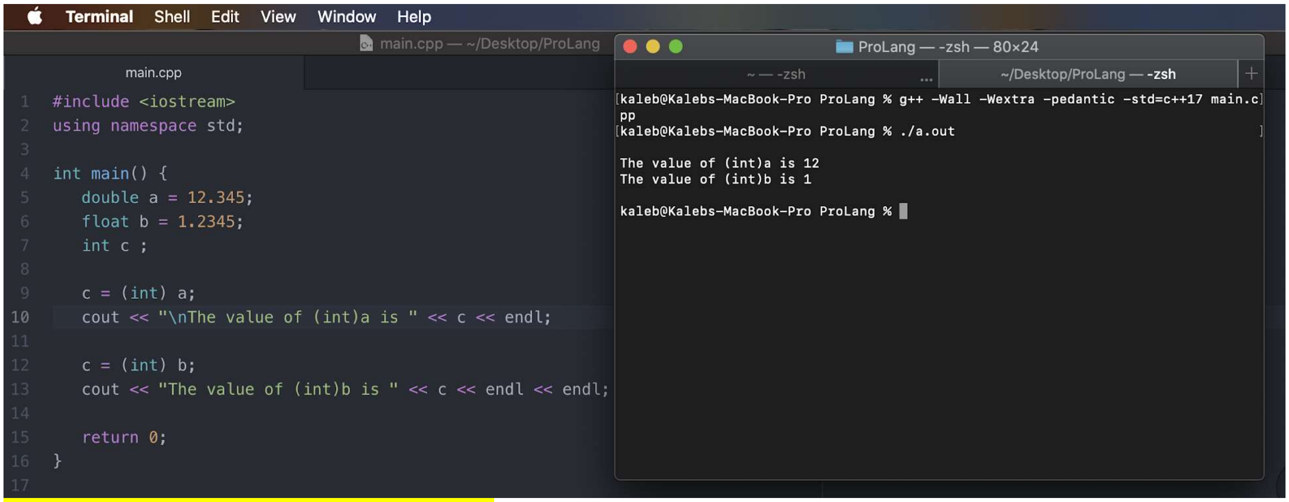
Kaleb Burdin
04/10/2020
Programming Languages



```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/testing.py ==============
Start int conversion
1
1
1
End int conversion

Start float conversion
1.0
1.1
1.0
1.1
End float conversion

Start str conversion
1
1
1.0
End str conversion
>>>
```

```python
x = int(1)   # x will be 1
y = int(1.1) # y will be 1
z = int("1") # z will be 1

print("Start int conversion")
print(x)
print(y)
print(z)
print("End int conversion\n")

x = float(1)     # x will be 1.0
y = float(1.1)   # y will be 1.1
z = float("1")   # z will be 1.0
w = float("1.1") # w will be 1.1

print("Start float conversion")
print(x)
print(y)
print(z)
print(w)
print("End float conversion\n")

x = str("1") # x will be '1'
y = str(1)   # y will be '1'
z = str(1.0) # z will be '1.0'

print("Start str conversion")
print(x)
print(y)
print(z)
print("End str conversion")
```

Example of type conversion in Python

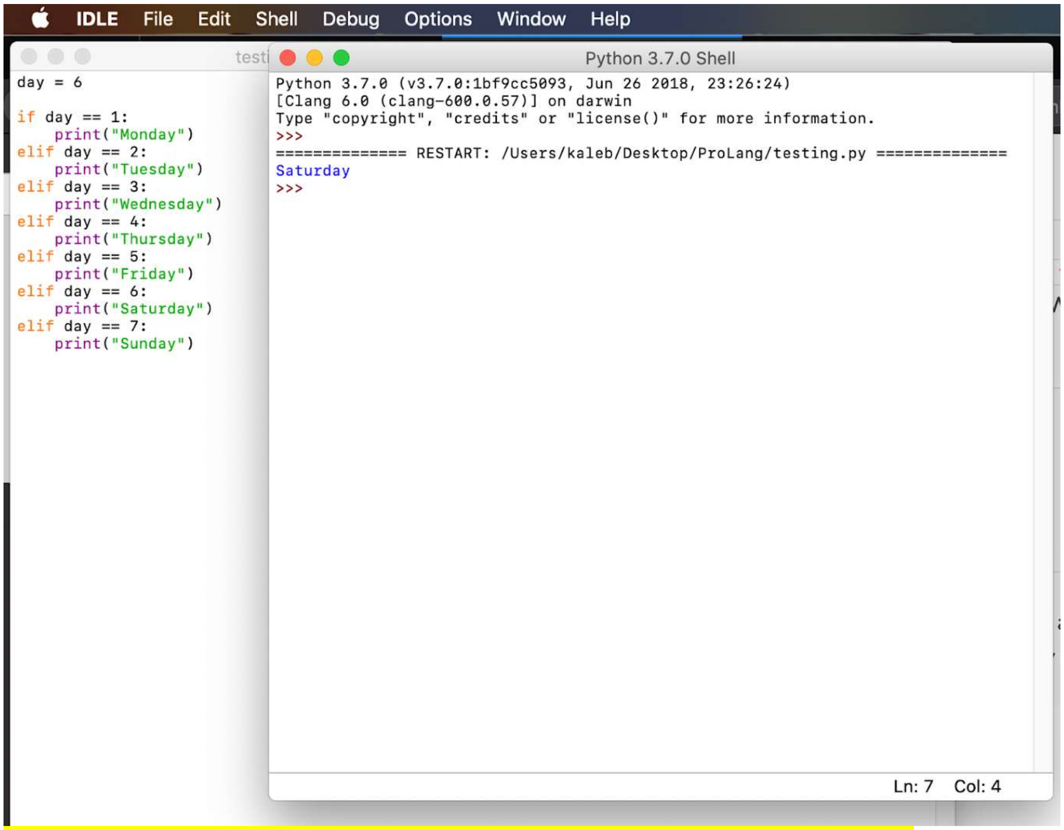


```cpp
#include <iostream>
using namespace std;

int main() {
   double a = 12.345;
   float b = 1.2345;
   int c ;

   c = (int) a;
   cout << "\nThe value of (int)a is " << c << endl;

   c = (int) b;
   cout << "The value of (int)b is " << c << endl << endl;

   return 0;
}
```

```
kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c
pp
kaleb@Kalebs-MacBook-Pro ProLang % ./a.out

The value of (int)a is 12
The value of (int)b is 1

kaleb@Kalebs-MacBook-Pro ProLang %
```

Example of type conversion in C++

Both Python and C++, as object-oriented languages, make use of selection control statements. In Python, decisions are made with the *if* statement, also known as the selection statement. When processing an *if* statement, the computer first evaluates some criterion or condition. If it is met, the specified action is performed. An optional part of an if statement is the *else* clause. It allows the programmer to specify an alternative instruction (or set of instructions) to be executed if the condition is *not* met. If two alternatives need to be handled then the elif statement is used in place of an *if/else* to improve readability. In order to understand the truthiness of these alternatives, conditional statements in Python make use of *booleans*. Conditions which consist of simpler conditions joined together with AND, OR, and NOT are referred to as *compound* *conditions*. These operators are known as *boolean operators*. C++ and Python are nearly identical in everything they do due to their object-oriented focus, thus C++, like Python, uses *if/else* statements and *elif* statements as selection control statements. C++ also makes use of the boolean operators, *&&* and ||, in the same sense that Python used AND, OR, AND NOT. to C++ also uses a thing called *switch* statements. A switch statement allows the program to select one of many blocks to be executed. The switch expression is evaluated once, and the value of the expression is compared with the values of each case. Just like Python and C++ essentially share the same selection control statements, they also share the same looping control structures.

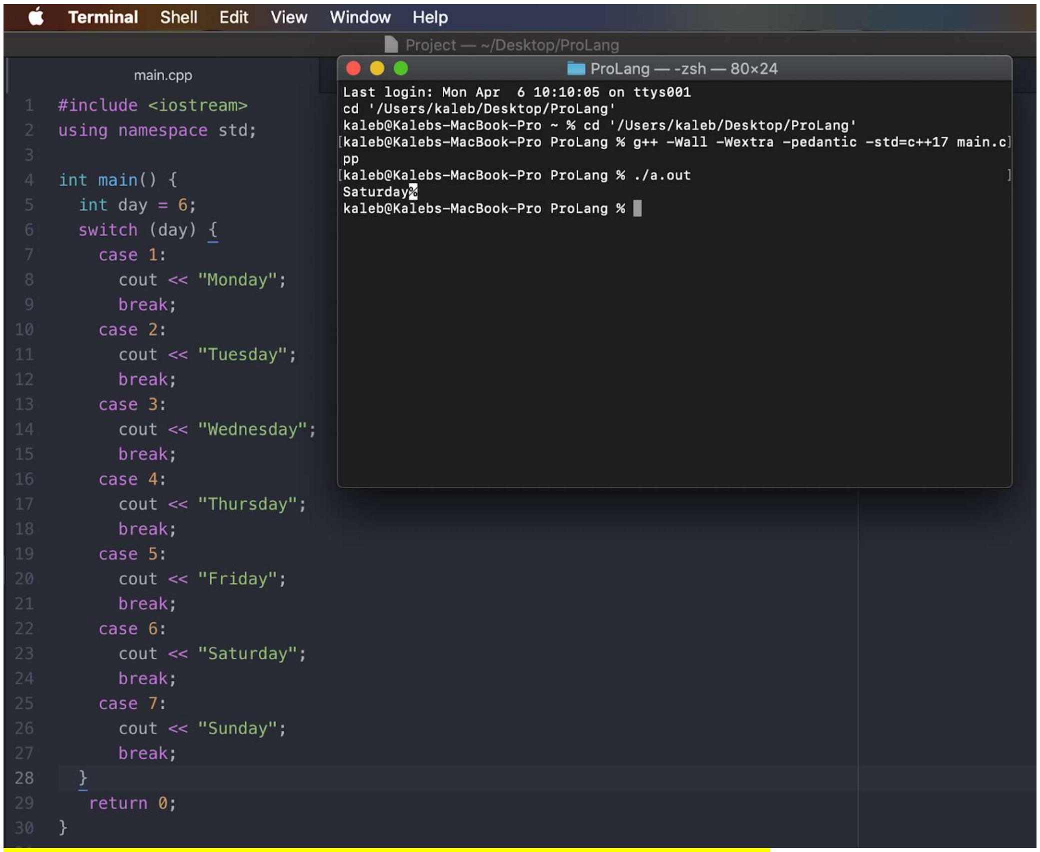


```
day = 6

if day == 1:
    print("Monday")
elif day == 2:
    print("Tuesday")
elif day == 3:
    print("Wednesday")
elif day == 4:
    print("Thursday")
elif day == 5:
    print("Friday")
elif day == 6:
    print("Saturday")
elif day == 7:
    print("Sunday")
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/testing.py ==============
Saturday
>>>
```

*Example of the *if/else* and *elif* selection control statements in Python*

Kaleb Burdin
04/10/2020
Programming Languages



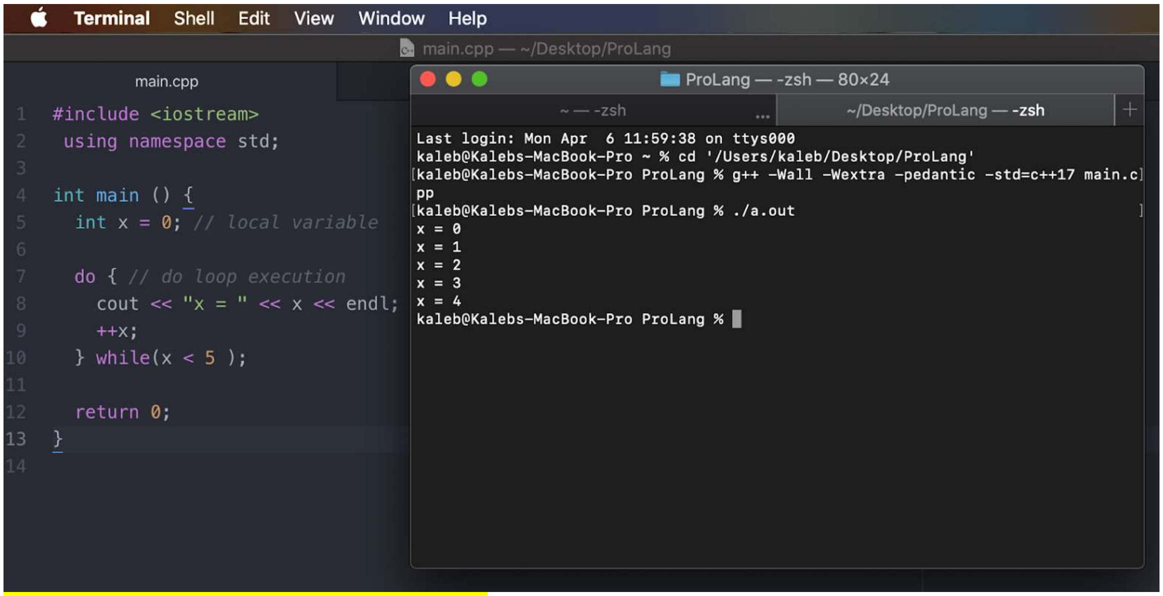
```cpp
#include <iostream>
using namespace std;

int main() {
  int day = 6;
  switch (day) {
    case 1:
      cout << "Monday";
      break;
    case 2:
      cout << "Tuesday";
      break;
    case 3:
      cout << "Wednesday";
      break;
    case 4:
      cout << "Thursday";
      break;
    case 5:
      cout << "Friday";
      break;
    case 6:
      cout << "Saturday";
      break;
    case 7:
      cout << "Sunday";
      break;
  }
   return 0;
}
```

```
Last login: Mon Apr  6 10:10:05 on ttys001
cd '/Users/kaleb/Desktop/ProLang'
kaleb@Kalebs-MacBook-Pro ~ % cd '/Users/kaleb/Desktop/ProLang'
kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c
pp
kaleb@Kalebs-MacBook-Pro ProLang % ./a.out
Saturday%
kaleb@Kalebs-MacBook-Pro ProLang % 
```

*Example of the *switch* selection control statement in C++*

Both Python and C++ make great use of looping control structures such as the *while* loop and *for* loop. While Python and C++ share the same while loop, Python does not provide the same for loop like other languages do. Python gives the programmer a *for/in* control structure- it's similar to the *for/each* control structure in other languages- in which it can be used to iterate over iterators and a range; this includes lists, strings, tuples, dictionaries, etc. C++ has its own little unique loop control statement called a *do/while* loop. A do/while loop enables the program to loop once whether the data is runnable in the program or not. In order for these loop control structures to work, though, the program needs to be able to group the appropriate code together in order to effectively execute the code. This can be done by defining a *code block*.

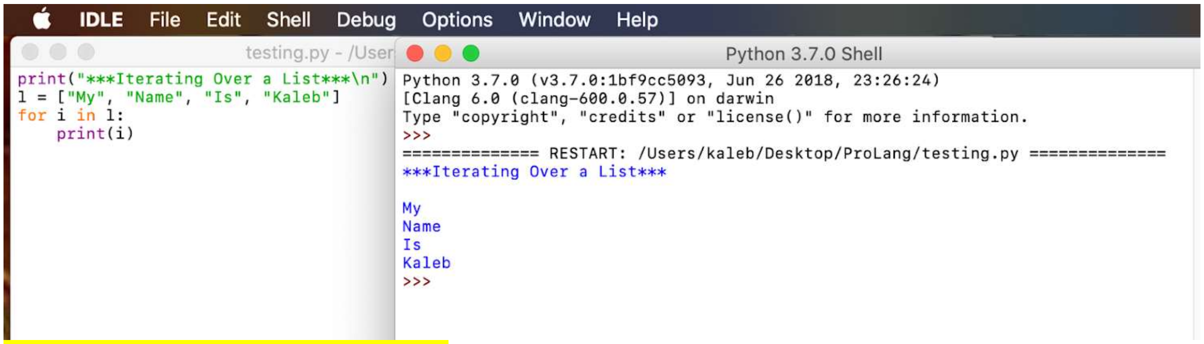
Kaleb Burdin
04/10/2020
Programming Languages



```
#include <iostream>
 using namespace std;

int main () {
  int x = 0; // local variable

  do { // do loop execution
    cout << "x = " << x << endl;
    ++x;
  } while(x < 5 );

  return 0;
}
```

```
Last login: Mon Apr  6 11:59:38 on ttys000
kaleb@Kalebs-MacBook-Pro ~ % cd '/Users/kaleb/Desktop/ProLang'
[kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c]
pp
[kaleb@Kalebs-MacBook-Pro ProLang % ./a.out                                      ]
x = 0
x = 1
x = 2
x = 3
x = 4
kaleb@Kalebs-MacBook-Pro ProLang %
```

*Example of a *do/while* loop in C++*



```
print("***Iterating Over a List***\n")
l = ["My", "Name", "Is", "Kaleb"]
for i in l:
    print(i)
```

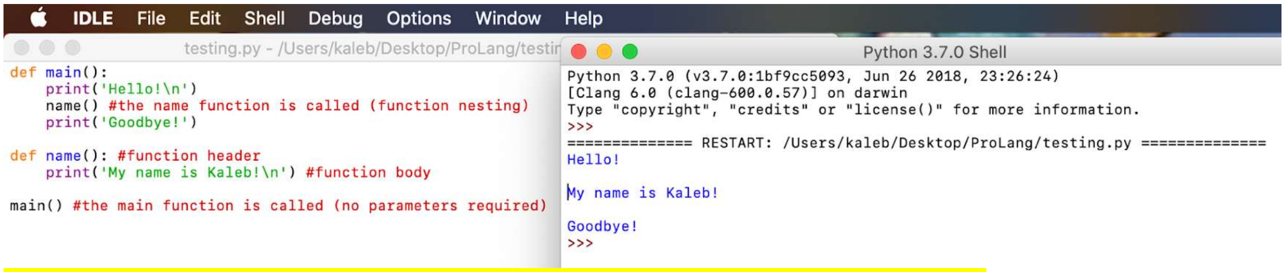
```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/testing.py ==============
***Iterating Over a List***

My
Name
Is
Kaleb
>>>
```

*Example of a *for/in* loop in C++*

In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements. C++, on the other hand, uses curly braces to define a code block and thus grouping statements. C++, also though, can do without curly braces in certain situations. Say you have an if statement in your C++ code that just increments a value and nothing else, there is only one line of code inside of that control structure then you can just indent the increment statement and that's it. So, in C++ you can just have an indented code block instead of curly braces if there's a single line of code in a structure.

Subprograms Comparison

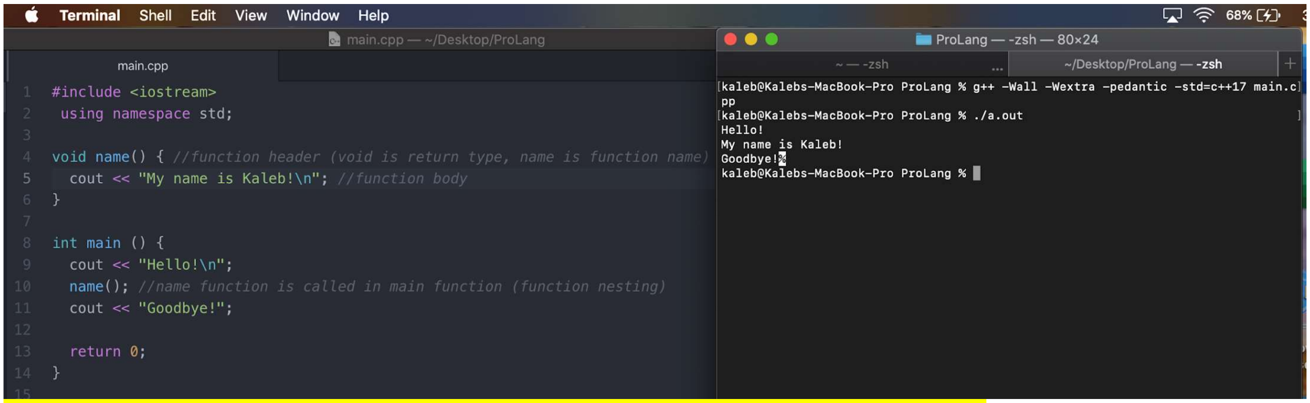
A function is a reusable portion of a program, sometimes called a *procedure* or *subroutine.* Functions are like mini programs (or subprograms) which can take in special inputs known as *arguments*, can produce an answer value known as a *return value*, and are similar to the idea of a function in mathematics. In both Python and C++, the basic structure for subprograms include a function header- which consists of the function return type, the function name, the formal parameter list, and any inheritance-, the function body- which contains a collection of statements that defines what the function does-, and a return inside the function body. Unlike C++, which requires a return type in the function header, Python includes the keyword *def* instead which allows the function to return any data type no matter what. A function definition specifies what a function does but needs to be called to execute it. Calling functions is identical in both C++ and Python in which the function name is called with all actual parameters required for the function to work. Many functions can exist together in which subprograms can be nested or called inside one another. Both C++ and Python support subprogram nesting by calling one function inside of another.



```
def main():
    print('Hello!\n')
    name() #the name function is called (function nesting)
    print('Goodbye!')

def name(): #function header
    print('My name is Kaleb!\n') #function body

main() #the main function is called (no parameters required)
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/testing.py ==============
Hello!

My name is Kaleb!

Goodbye!
>>>
```

Example of defining and calling a function and function nesting in Python



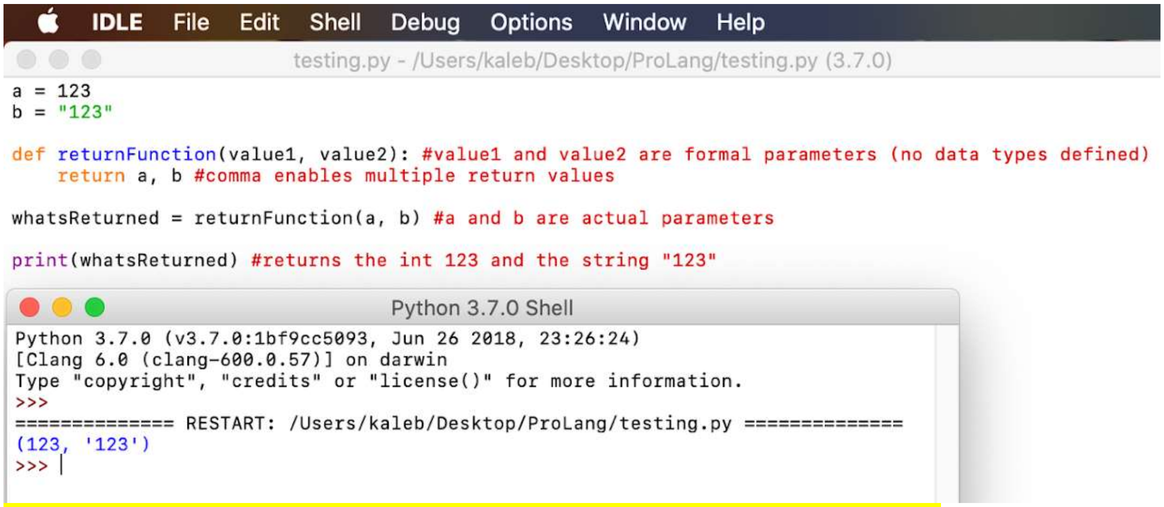
```
#include <iostream>
 using namespace std;

void name() { //function header (void is return type, name is function name)
  cout << "My name is Kaleb!\n"; //function body
}

int main () {
  cout << "Hello!\n";
  name(); //name function is called in main function (function nesting)
  cout << "Goodbye!";

  return 0;
}
```

```
[kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c]
pp
[kaleb@Kalebs-MacBook-Pro ProLang % ./a.out                                      ]
Hello!
My name is Kaleb!
Goodbye!%
kaleb@Kalebs-MacBook-Pro ProLang % 
```

Example of defining and calling a function and function nesting in Python

All functions in all languages require some sort of parameter or special input to be passed in in order to manipulate. Typically, there are two types of parameters: formal parameters- identifiers used in the function definition to represent corresponding actual arguments-, and actual parameters- values (or variables)/expressions that are used inside the parentheses of a function call. Formal parameters are defined nearly identically in both C++ and Python, but just like the function headers for both languages, there's a slight difference. C++ requires that a data type (i.e. int, string, etc.) be defined for each formal parameter name while Python does not require the data type and only the formal parameter name. Both of these languages also differ in the way that values are returned from a function. As said before, function headers in C++ require a return type (i.e. void, int, string, etc.) to be stated while Python function headers only require the keyword *def* which allows any data type to be returned from the function. So, in short, the return type in C++ is specified in the function header while Python does not specify a return type anywhere. The value wanting to be returned is returned in C++ and Python through the keyword *return* inside of the function body. In Python, you can return multiple values by simply returning them separated by commas (i.e. return 123, "123" returns the int 123 and the string "123"). C++, though, does not currently provide a way to return multiple values from a single return statement.

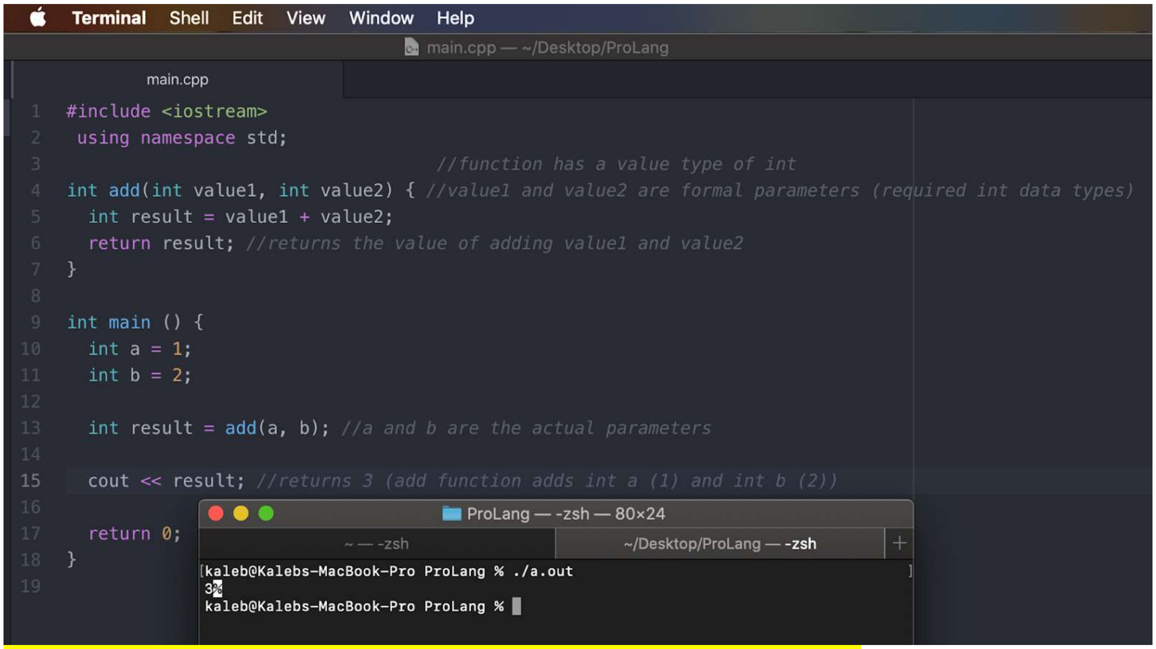
Kaleb Burdin
04/10/2020
Programming Languages


IDLE File Edit Shell Debug Options Window Help
testing.py - /Users/kaleb/Desktop/ProLang/testing.py (3.7.0)

```python
a = 123
b = "123"

def returnFunction(value1, value2): #value1 and value2 are formal parameters (no data types defined)
    return a, b #comma enables multiple return values

whatsReturned = returnFunction(a, b) #a and b are actual parameters

print(whatsReturned) #returns the int 123 and the string "123"
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/testing.py ==============
(123, '123')
>>> |
```

Example of formal parameters and returning multiple values in Python

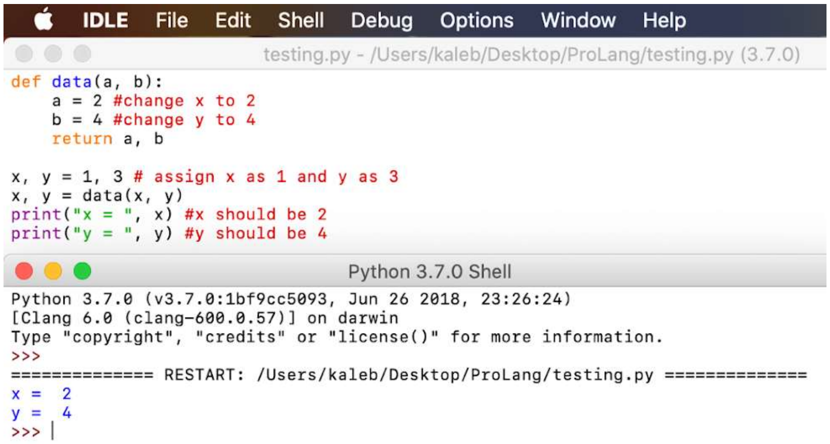

Terminal Shell Edit View Window Help
main.cpp — ~/Desktop/ProLang
main.cpp

```cpp
#include <iostream>
 using namespace std;
                                   //function has a value type of int
int add(int value1, int value2) { //value1 and value2 are formal parameters (required int data types)
  int result = value1 + value2;
  return result; //returns the value of adding value1 and value2
}

int main () {
  int a = 1;
  int b = 2;

  int result = add(a, b); //a and b are the actual parameters

  cout << result; //returns 3 (add function adds int a (1) and int b (2))

  return 0;
}
```

```
[kaleb@Kalebs-MacBook-Pro ProLang % ./a.out
3%
kaleb@Kalebs-MacBook-Pro ProLang %
```

Example of formal parameters and returning an int value in C++

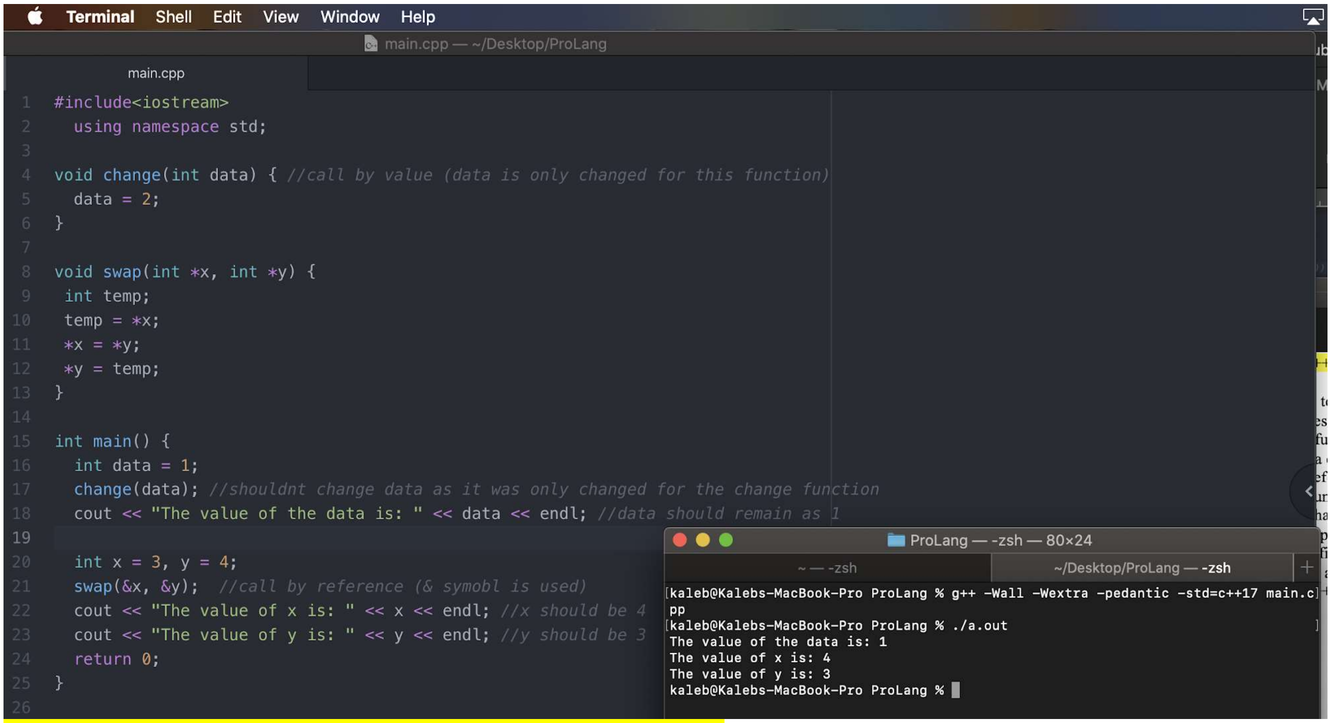
The most common evaluation strategy when passing arguments to a function has been call by value and call by reference. In call by value, the argument expression is evaluated, and the result of this evaluation is bound to the corresponding variable in the function. In call by reference, a function gets an implicit reference to the argument, rather than a copy of its value. Python, though, utilizes a system, which is known as “Call by Object Reference” or “Call by assignment”. In the event that you pass arguments like whole numbers, strings, or tuples to a function, the passing is like call-by-value because you cannot change the value of the immutable objects being passed to the function. C++, on the other hand, implements both call by value and call be reference. In call by value, the **original value is not modified while** in call by reference, original value is modified because the address is passed. In C++ and Python, call by value is used by using the default parameter implementation while in C++, call by reference requires that the ‘*&*’ symbol is added before the parameter name.


IDLE File Edit Shell Debug Options Window Help
testing.py - /Users/kaleb/Desktop/ProLang/testing.py (3.7.0)
Python 3.7.0 Shell

```python
def data(a, b):
    a = 2 #change x to 2
    b = 4 #change y to 4
    return a, b

x, y = 1, 3 # assign x as 1 and y as 3
x, y = data(x, y)
print("x = ", x) #x should be 2
print("y = ", y) #y should be 4
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/testing.py ==============
x =  2
y =  4
>>> |
```

example of call by object reference in Python


Terminal Shell Edit View Window Help
main.cpp — ~/Desktop/ProLang
main.cpp
ProLang — -zsh — 80×24
~ — -zsh
~/Desktop/ProLang — -zsh

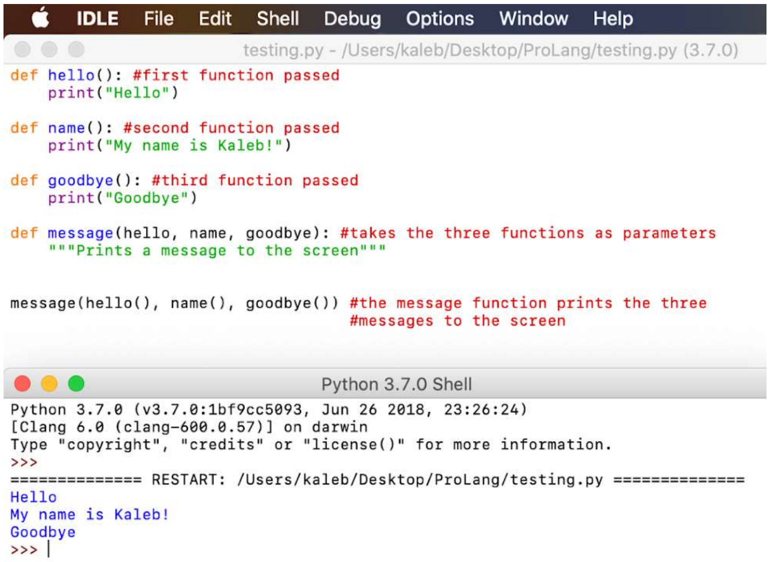
```cpp
#include<iostream>
  using namespace std;

void change(int data) { //call by value (data is only changed for this function)
  data = 2;
}

void swap(int *x, int *y) {
 int temp;
 temp = *x;
 *x = *y;
 *y = temp;
}

int main() {
  int data = 1;
  change(data); //shouldnt change data as it was only changed for the change function
  cout << "The value of the data is: " << data << endl; //data should remain as 1

  int x = 3, y = 4;
  swap(&x, &y);  //call by reference (& symobl is used)
  cout << "The value of x is: " << x << endl; //x should be 4
  cout << "The value of y is: " << y << endl; //y should be 3
  return 0;
}
```

```
[kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c
pp
[kaleb@Kalebs-MacBook-Pro ProLang % ./a.out
The value of the data is: 1
The value of x is: 4
The value of y is: 3
kaleb@Kalebs-MacBook-Pro ProLang % ▉
```

example of call by value and call by reference in C++

One helpful aspect of Python is that programmers can pass functions into other functions. Function can be passed around because in Python, functions are objects. If you don't want to name a new function each time, you will have the option to pass an anonymous function (i.e.: *lambda*) instead. C++ also enables the programmer to pass functions as parameters too, but in a different aspect. You must pass the return type, the ‘*’ symbol, the function name directly afterwards, and then any parameter of the function being passed as a parameter.

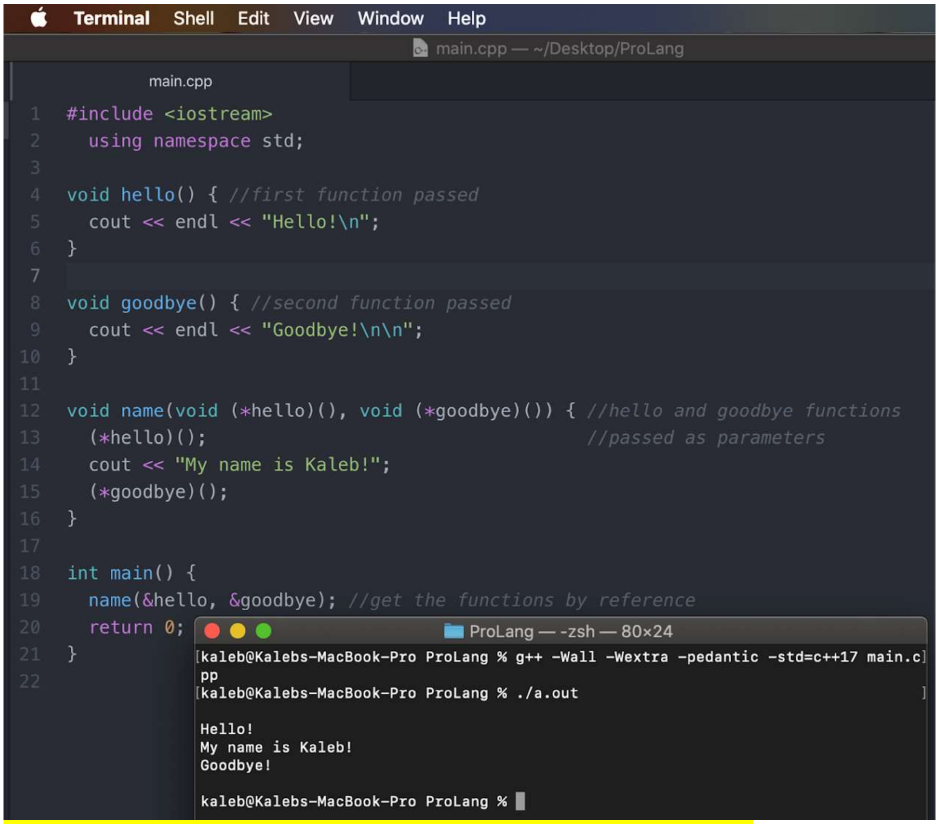
Kaleb Burdin
04/10/2020
Programming Languages


IDLE File Edit Shell Debug Options Window Help
testing.py - /Users/kaleb/Desktop/ProLang/testing.py (3.7.0)
Python 3.7.0 Shell

```python
def hello(): #first function passed
    print("Hello")

def name(): #second function passed
    print("My name is Kaleb!")

def goodbye(): #third function passed
    print("Goodbye")

def message(hello, name, goodbye): #takes the three functions as parameters
    """Prints a message to the screen"""


message(hello(), name(), goodbye()) #the message function prints the three
                                    #messages to the screen
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/testing.py ==============
Hello
My name is Kaleb!
Goodbye
>>> |
```

example of functions being passed as arguments in Python

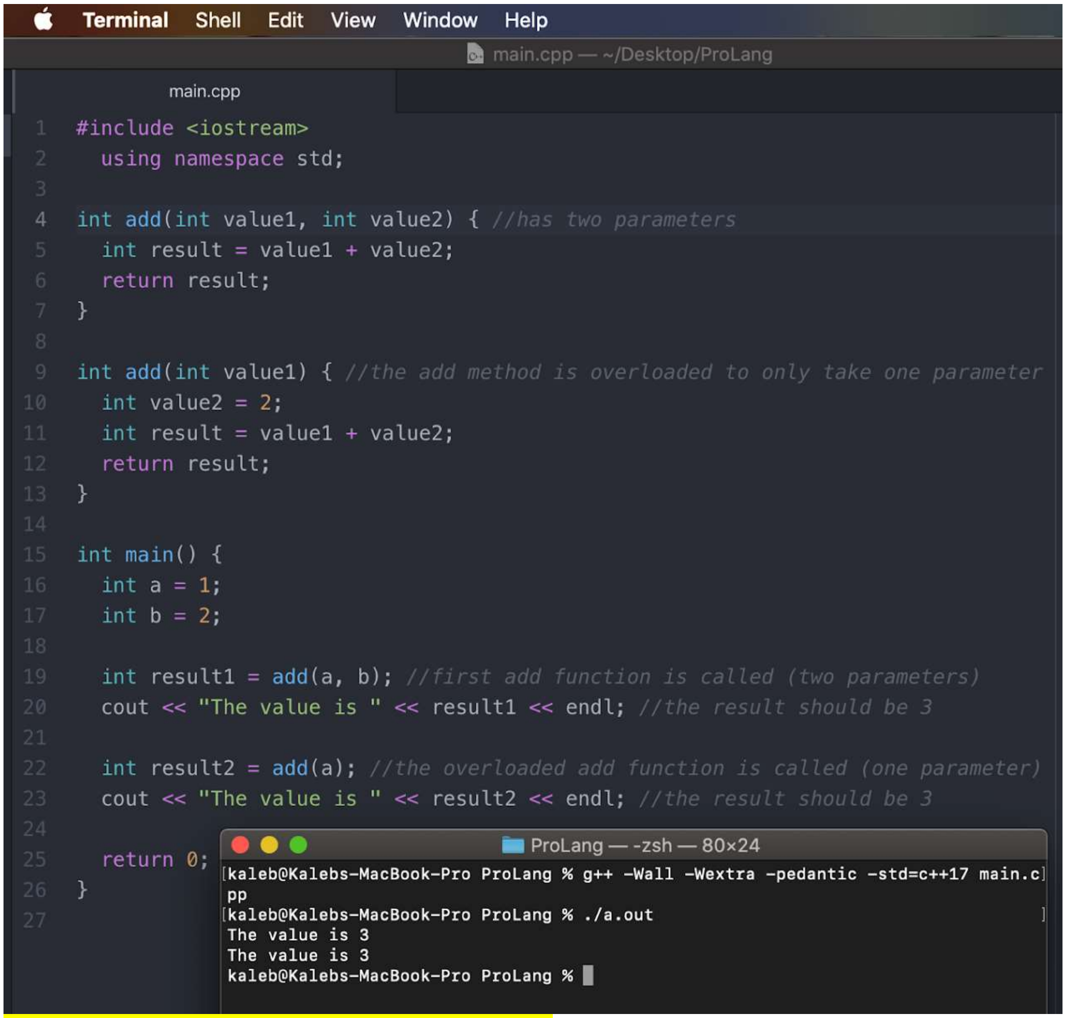

Terminal Shell Edit View Window Help
main.cpp — ~/Desktop/ProLang
main.cpp
ProLang — -zsh — 80×24

```cpp
#include <iostream>
  using namespace std;

void hello() { //first function passed
  cout << endl << "Hello!\n";
}

void goodbye() { //second function passed
  cout << endl << "Goodbye!\n\n";
}

void name(void (*hello)(), void (*goodbye)()) { //hello and goodbye functions
  (*hello)();                                   //passed as parameters
  cout << "My name is Kaleb!";
  (*goodbye)();
}

int main() {
  name(&hello, &goodbye); //get the functions by reference
  return 0;
}
```

```
kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c
pp
kaleb@Kalebs-MacBook-Pro ProLang % ./a.out

Hello!
My name is Kaleb!
Goodbye!

kaleb@Kalebs-MacBook-Pro ProLang % 
```

example of functions being passed as arguments in C++

Function overloading is the ability to have multiple functions with the same name but with different signatures/implementations. When an overloaded function is called, the runtime first evaluates the arguments/parameters passed to the function call and judging by this invokes the corresponding implementation. Python does not support function overloading. When we define multiple functions with the same name, the later one always overrides the prior and thus, in the namespace, there will always be a single entry against each function name. C++, though, does support function overloading in which many separate functions can host the same name, but they have to have different parameter lists.



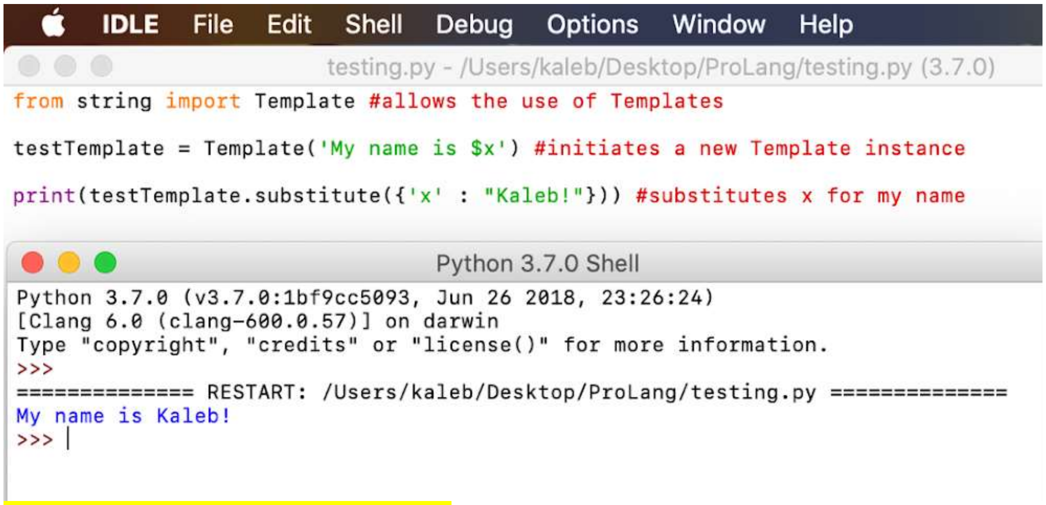
```cpp
#include <iostream>
  using namespace std;

int add(int value1, int value2) { //has two parameters
  int result = value1 + value2;
  return result;
}

int add(int value1) { //the add method is overloaded to only take one parameter
  int value2 = 2;
  int result = value1 + value2;
  return result;
}

int main() {
  int a = 1;
  int b = 2;

  int result1 = add(a, b); //first add function is called (two parameters)
  cout << "The value is " << result1 << endl; //the result should be 3

  int result2 = add(a); //the overloaded add function is called (one parameter)
  cout << "The value is " << result2 << endl; //the result should be 3

  return 0;
}
```

```
[kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c]
pp
[kaleb@Kalebs-MacBook-Pro ProLang % ./a.out                                       ]
The value is 3
The value is 3
kaleb@Kalebs-MacBook-Pro ProLang % 
```

example of function overloading in C++

Templates are powerful features which allow programmers to write generic programs. In simple terms, a programmer can create a single function or a class to work with different data types using templates. Both C++ and Python support *Templates* but in their own special ways. In C++, there are two types of *Templates*: function templates and class templates. A function template starts with the keyword *template* followed by template parameters inside <> which is followed by function declaration (i.e. template <class Name>). In Python, a template is a class of String module; it allows for data to change without having to edit the application. A template class takes a string as a template, within the string use placeholder variable name preceding the '$'

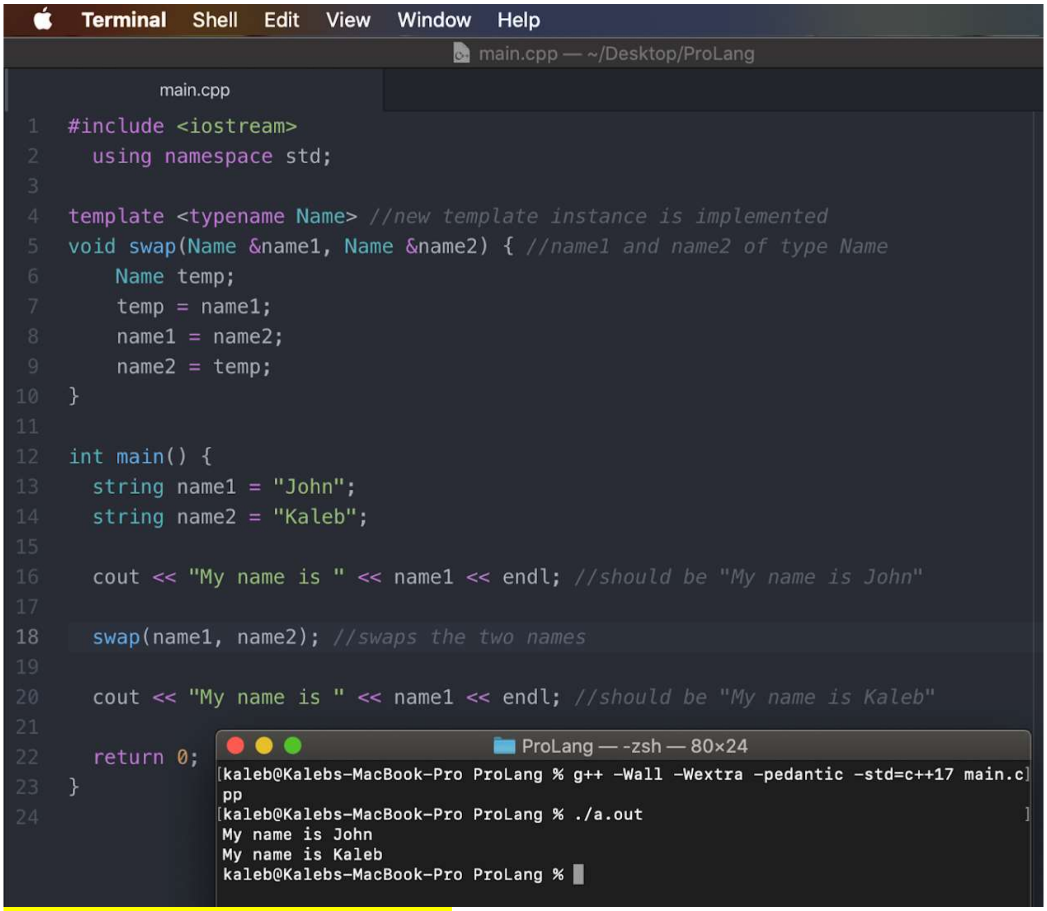
symbol to depict where the placeholder is. Templates are implemented by assigning a variable to Template followed by a string of characters.


IDLE
File
Edit
Shell
Debug
Options
Window
Help
testing.py - /Users/kaleb/Desktop/ProLang/testing.py (3.7.0)
Python 3.7.0 Shell

```python
from string import Template #allows the use of Templates

testTemplate = Template('My name is $x') #initiates a new Template instance

print(testTemplate.substitute({'x' : "Kaleb!"})) #substitutes x for my name
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/testing.py ==============
My name is Kaleb!
>>> |
```

example of Templates in Python



```cpp
#include <iostream>
  using namespace std;

template <typename Name> //new template instance is implemented
void swap(Name &name1, Name &name2) { //name1 and name2 of type Name
    Name temp;
    temp = name1;
    name1 = name2;
    name2 = temp;
}

int main() {
  string name1 = "John";
  string name2 = "Kaleb";

  cout << "My name is " << name1 << endl; //should be "My name is John"

  swap(name1, name2); //swaps the two names

  cout << "My name is " << name1 << endl; //should be "My name is Kaleb"

  return 0;
}
```

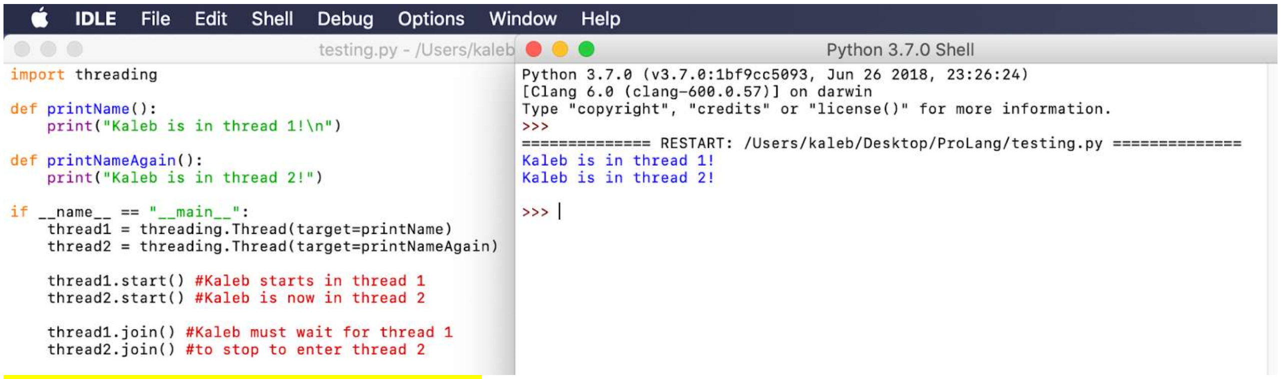
```
[kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c]
pp
[kaleb@Kalebs-MacBook-Pro ProLang % ./a.out                                      ]
My name is John
My name is Kaleb
kaleb@Kalebs-MacBook-Pro ProLang % █
```

example of Templates in Python

Special Features Comparison

Kaleb Burdin
04/10/2020
Programming Languages

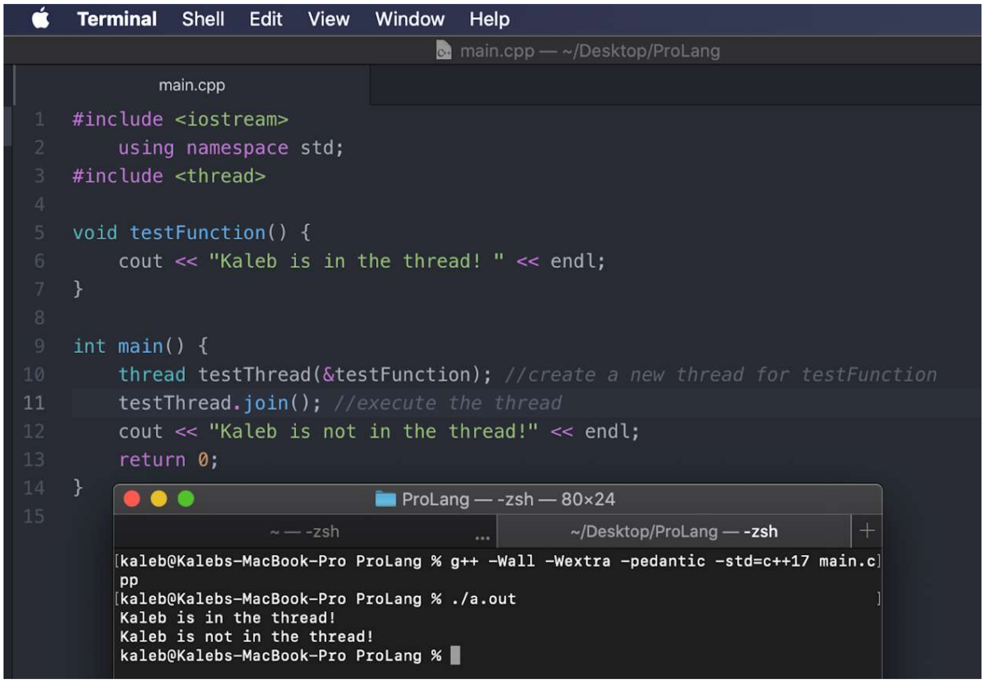
At the simplest and most basic level, concurrency is about two or more separate activities happening at the same time. Python does have built-in libraries for the most common concurrent programming constructs- multiprocessing and multithreading-, but multithreading in Python is not really multithreading due to the global interpreter lock in Python. In Python, multi-threading is supported by the global interpreter lock which is a mutex. This is to prevent multiple threads from accessing the same Python object simultaneously. Only one thread can hold the global interpreter lock at a time, one thread must wait for another thread to release the global interpreter lock before running which essentially removes the functionality of multithreading. In a single-threaded C++ program, execution starts at main(), and then proceeds in a sequential fashion. In a multi-threaded program, the first thread starts at main, but additional threads may be started by the application which start at a user-specified function. These then run concurrently, or in parallel with the original thread. The creation of concurrent elements, a synchronization mechanism, and mutual exclusion are all implicitly encapsulated with an object or the class definition for languages in the first category.



```python
import threading

def printName():
    print("Kaleb is in thread 1!\n")

def printNameAgain():
    print("Kaleb is in thread 2!")

if __name__ == "__main__":
    thread1 = threading.Thread(target=printName)
    thread2 = threading.Thread(target=printNameAgain)

    thread1.start() #Kaleb starts in thread 1
    thread2.start() #Kaleb is now in thread 2

    thread1.join() #Kaleb must wait for thread 1
    thread2.join() #to stop to enter thread 2
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/testing.py ==============
Kaleb is in thread 1!
Kaleb is in thread 2!

>>> |
```

Example of concurrency in Python



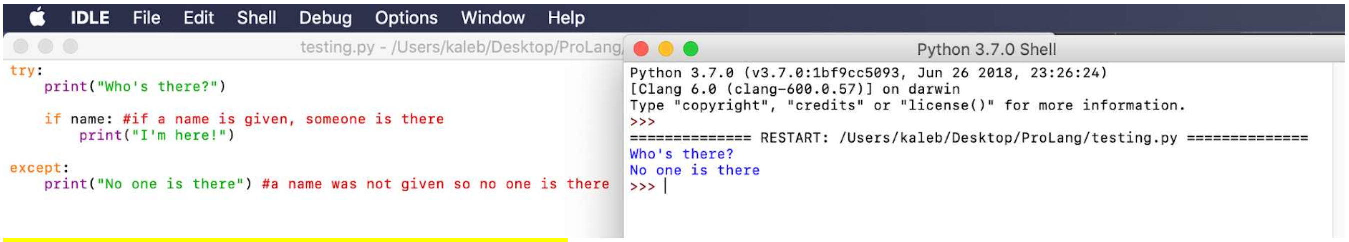
```cpp
#include <iostream>
    using namespace std;
#include <thread>

void testFunction() {
    cout << "Kaleb is in the thread! " << endl;
}

int main() {
    thread testThread(&testFunction); //create a new thread for testFunction
    testThread.join(); //execute the thread
    cout << "Kaleb is not in the thread!" << endl;
    return 0;
}
```

```
[kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c]
pp
[kaleb@Kalebs-MacBook-Pro ProLang % ./a.out                                      ]
Kaleb is in the thread!
Kaleb is not in the thread!
kaleb@Kalebs-MacBook-Pro ProLang % █
```

Kaleb Burdin
04/10/2020
Programming Languages

Example of concurrency in C++

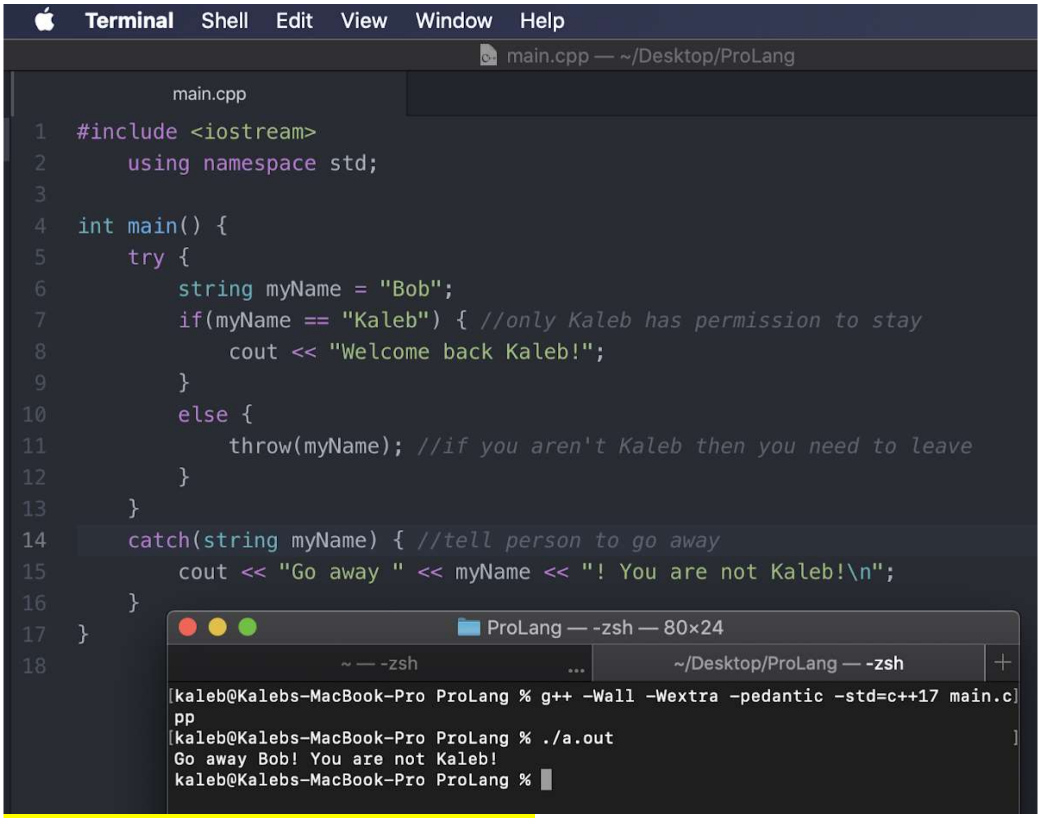
Both Python and C++ support the use of exception handling. In Python, the programmer can raise an exception by using the **raise** keyword which throws an exception if a certain condition occurs (i.e. raise Exception('wrong data type used')). Python also uses the **assert** keyword to keep from having to wait for the program to crash midway. The **assert** keyword ensures that a certain condition is met; if the condition turns out to be True then the program can continue but if the condition turns out to be False then the program can throw an exception. The *try* and *except* block in the language used to catch and handle exceptions. Everything after the **try** keyword is read as a normal part of the program while the code after the **except** keyword is the response to the *try statement*. The *except statement* determines how your program responds to exceptions and can be handled in many different ways. Python can either print a message to the user about what the problem is or it can use the **pass** keyword to prevent the program from crashing without telling the user what the problem is. The **else** keyword is also used by the language to execute code when no exceptions have occurred. Lastly, the **finally** keyword to execute sections of code that should run no matter what, with or without any previously encountered exceptions. C++ uses some of the same techniques as Python to handle exceptions. C++ uses three different keywords in exception handling: **throw**, **catch**, and **try**. The **try** and **catch** keywords work the same as the **try** and **except** keywords in Python. The **throw** keyword, though, is new as it can be used to throw an exception anywhere within a code block. So, the other keywords are not necessary for exception handling.



```
try:
    print("Who's there?")

    if name: #if a name is given, someone is there
        print("I'm here!")

except:
    print("No one is there") #a name was not given so no one is there
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
============== RESTART: /Users/kaleb/Desktop/ProLang/testing.py ==============
Who's there?
No one is there
>>> |
```

Example of exception handling in Python



```cpp
#include <iostream>
    using namespace std;

int main() {
    try {
        string myName = "Bob";
        if(myName == "Kaleb") { //only Kaleb has permission to stay
            cout << "Welcome back Kaleb!";
        }
        else {
            throw(myName); //if you aren't Kaleb then you need to leave
        }
    }
    catch(string myName) { //tell person to go away
        cout << "Go away " << myName << "! You are not Kaleb!\n";
    }
}
```

```
[kaleb@Kalebs-MacBook-Pro ProLang % g++ -Wall -Wextra -pedantic -std=c++17 main.c]
pp
[kaleb@Kalebs-MacBook-Pro ProLang % ./a.out                                       ]
Go away Bob! You are not Kaleb!
kaleb@Kalebs-MacBook-Pro ProLang % 
```

Example of exception handling in C++

When it comes to whether Python and C++ are purely functional languages, the answer is both yes and no. Python supports a functional style of programming by using higher order functions, using comprehensions, and using generators as lazily evaluated lists. It is possible to be purely functional when coding in Python due to some of the features said above but in general, Python is not purely functional as it allows the programmer to create and use impure functions and mutable data structures in which a purely functional language requires all functions to be pure and all data structures to be immutable so data isn't changed somehow. C++ is in the same boat as Python in how the programmer can write in an almost pure functional style, but it has to be directly enforced by the program and has to go out of the way in which the language was originally designed. C++ requires the use of the **const** keyword in order to keep data constant or immutable. C++ was not meant to be used this way and thus readability will decrease between programmers and life will be a lot harder when actually coding. C++, though, like Python, does support some functional style programming in the way it uses template meta-programming. In C++, creating new types, functions, and classes via templates is done through a Turing complete template language which has to be functional in order to compile.

Conclusion

Python and C++ certainly are popular programming languages, but they aren't the only ones. There are thousands of programming languages in the world that mimic the English language in order to effectively communicate functionality to the computing machines of today. Due to both of the languages being object-oriented, they're more alike than, say, a language that is object-oriented and one that isn't. The comparison between these two languages hopefully demonstrates the difference in design between two languages and how each and every language is able to have its own little unique implementation.

Resources Used

Python

https://www.tutorialsteacher.com/python/python-data-types

https://jpt-pynotes.readthedocs.io/en/latest/scalar-types.html

https://www.tutorialspoint.com/python/python_classes_objects.htm

http://www.dalkescientific.com/writings/NBN/python_intro/literals.html

https://realpython.com/python-variables/#variable-assignment

https://stackoverflow.com/questions/11328920/is-python-strongly-typed

https://www.w3schools.com/python/python_scope.asp https://www.datacamp.com/community/tutorials/functions-python-tutorial

C++

https://www.geeksforgeeks.org/c-data-types/

https://stackoverflow.com/questions/6623130/scalar-vs-primitive-data-type-are-they-the-same-thing

https://www.geeksforgeeks.org/c-data-types/

https://www.tutorialspoint.com/cplusplus/cpp_constants_literals.htm

https://www.geeksforgeeks.org/variables-in-c/

https://stackoverflow.com/questions/26753483/is-c-considered-weakly-typed-why

https://www.geeksforgeeks.org/scope-of-variables-in-c/

https://www.tutorialspoint.com/cplusplus/cpp_functions.htm