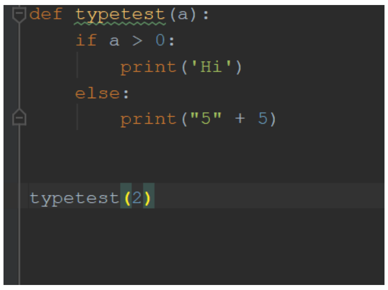
# Python VS C#

Python and C# are both very popular and widely used languages. Both languages are used for object-oriented programming however C# is a much stricter OOP language and python allows for some procedural programming. Both languages share many similarities however there are a few key differences which I will be discussing. The major difference that will be discussed in this report is the difference between static typing and dynamic typing as well as compiled vs interpreted languages.

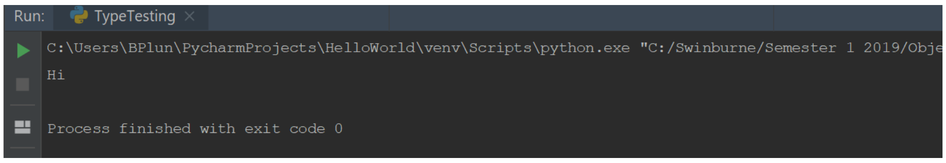
C# is a statically compiled language and Python is a dynamically interpreted language.  First let's look at the difference between complied and interpreted. Python is an interpreted language this means that the code is executed and at run time and is each line is parsed into an interpreter which then executes the commands. This means that the source code essentially is the program and allows for portability as to run the program only the specific interpreter is required. On the other hand, C# is a compiled program meaning that the code is compiled into a program that is written in assembly language which is then converted to binary by an assembler for the architecture of the device that the code was written on. This in turn means that complied languages are much less portable as running code on a machine that it was not written for may not work.

Along with this there are other differences between an interpreted and a compiled language. This includes the better performance provided by compiled language because the program is generated in its entirety before it is run as opposed to being generated while the code is being executes. However interpreted languages tend to have a faster development time as the rules are not as strict as they may be for a compiled language.

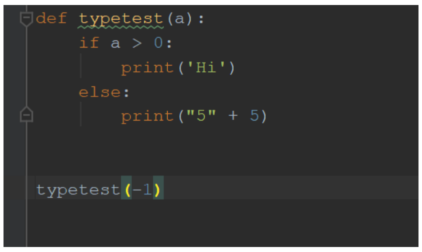
Next let's look at the difference between static and dynamically type languages. Dynamic type checking means that code is only type checked on execution, for example if the code "5"+5 was written in the code but not executed it would not cause an error, however if it is executed it will throw an exception as shown bey the example below.



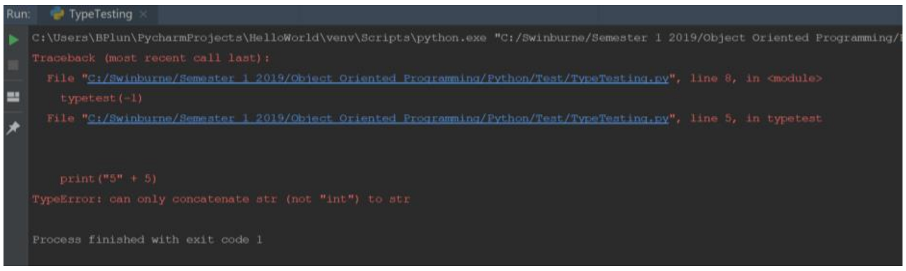
```
def typetest(a):
    if a > 0:
        print('Hi')
    else:
        print("5" + 5)


typetest(2)
```



```
C:\Users\BPlun\PycharmProjects\HelloWorld\venv\Scripts\python.exe "C:/Swinburne/Semester 1 2019/Obje
Hi

Process finished with exit code 0
```



```
def typetest(a):
    if a > 0:
        print('Hi')
    else:
        print("5" + 5)


typetest(-1)
```

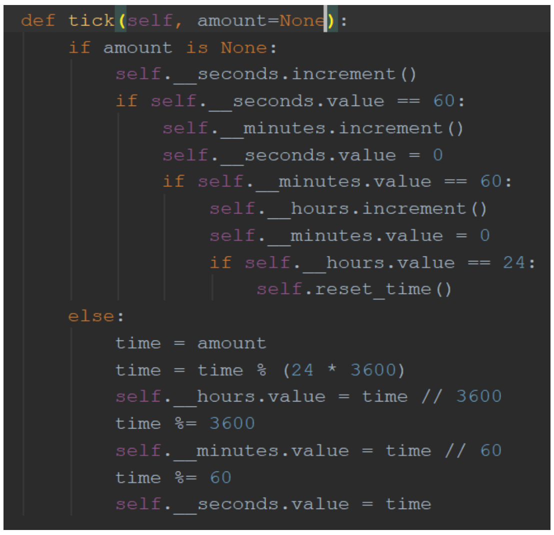


```
C:\Users\BPlun\PycharmProjects\HelloWorld\venv\Scripts\python.exe "C:/Swinburne/Semester 1 2019/Object Oriented Programming/
Traceback (most recent call last):
  File "C:/Swinburne/Semester 1 2019/Object Oriented Programming/Python/Test/TypeTesting.py", line 8, in <module>
    typetest(-1)
  File "C:/Swinburne/Semester 1 2019/Object Oriented Programming/Python/Test/TypeTesting.py", line 5, in typetest


    print("5" + 5)
TypeError: can only concatenate str (not "int") to str

Process finished with exit code 1
```

We can see that in the first example the else statement was not looked at therefore the “5”+5 was never executed meaning no error was thrown, however in the second example when this code was executed it produced an error relating to the types of the data. In contrast to this Static typing is when all type checking for code is done when the program is compiled. Using the last example again the line “5” + 5 would produce an error before the program is even run as only strings can be concatenated. Even though this line may not be executed the static type checking means that the program will not be compiled.

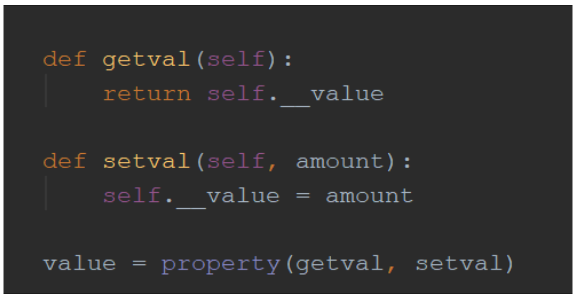
These key differences between the languages result in different advantages. With the use of C# and the fact that it is a static compiled language it means the program will perform better as it compiles and translates all the code before the program is run however it can take longer to develop programs as the rules are much stricter and there are many different conventions and intricacies that need to be learnt. On the other hand, although Python programs may have slower performance, Python generally has a faster development time due to the relatively straight forward nature of its syntax and conventions.

Python and C# also have some smaller differences in relation to their syntax. I noticed some of these differences whilst recreating the clock program in Python. The first major difference I noticed was in the creation of methods within a class. In Python when declaring a method, you must always pass in self as a parameter. Self refers to the object itself meaning properties and methods can be called from within the class, however in C# there is no need for this as it is assumed what you are referring to when calling methods and properties making it somewhat faster to create  new methods however it is much clearer what is happening in Python. Below is an example of this.



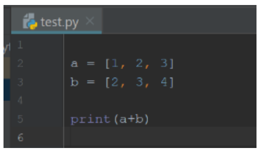
```python
def tick(self, amount=None):
    if amount is None:
        self.__seconds.increment()
        if self.__seconds.value == 60:
            self.__minutes.increment()
            self.__seconds.value = 0
            if self.__minutes.value == 60:
                self.__hours.increment()
                self.__minutes.value = 0
                if self.__hours.value == 24:
                    self.reset_time()
    else:
        time = amount
        time = time % (24 * 3600)
        self.__hours.value = time // 3600
        time %= 3600
        self.__minutes.value = time // 60
        time %= 60
        self.__seconds.value = time
```

Another key difference between Python and C# relating to the Object-oriented nature of the two languages is Pythons lack of the Private keyword. In C# Private is a keyword used in the process of encapsulation meaning only the object itself can access the field. Python relies on the convention of using not accessing a variable outside of the class if it has a underscore prefix and a somewhat private variable can be made with the use of the a double underscore prefix however it can still be accessed in contrast to C# in which using the keyword private prevents the variable form being accessed. This could be somewhat confusing and cause some issues when it comes to encapsulation. In saying that Python does utilise properties which are created to allow access of these “private” variables however the syntax differs slightly compared to C# which makes it a little more difficult.

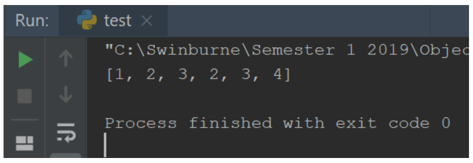


```python
def getval(self):
    return self.__value

def setval(self, amount):
    self.__value = amount

value = property(getval, setval)
```

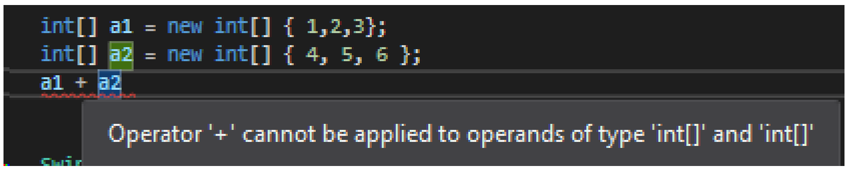
The differences of being a static or dynamically typed language play heavily into how Object-Oriented design differs between the two languages. One of the major examples of this is in the principle of Polymorphism and the design of methods as well as how objects are instantiated. Python has a focus on simple and readable code that can be understood and developed quickly and is the main reasoning behind the use of dynamic typing. Dynamic typing as mentioned previously removes the need for specification of variable types when creating things like class attributes, method Parameters, or instance variables. This has a big effect on how polymorphism fits into the design of a program. Not having the need to statically type variables means that along with having polymorphism in classes it can be seen everywhere through the program. Having un-typed variables means operations can be performed on a whole range of different variables and methods can support multiple different inputs. Operators such as ‘+’ can be polymorphic in that it can add many different variable types or objects which support the ‘add’ operation. Whereas in C# the static typing means some operators can only be used on objects of the same type. This is shown below.



```python
a = [1, 2, 3]
b = [2, 3, 4]

print(a+b)
```



```
"C:\Swinburne\Semester 1 2019\Objec
[1, 2, 3, 2, 3, 4]

Process finished with exit code 0
```



```csharp
int[] a1 = new int[] { 1,2,3};
int[] a2 = new int[] { 4, 5, 6 };
a1 + a2
```

```
Operator '+' cannot be applied to operands of type 'int[]' and 'int[]'
```

This impacts on how a program is designed in python as there is potential to destroy this polymorphism which I something that should be avoided. Performing type checks or using methods such as isinstance can remove this polymorphism. Having this dynamic type checking means that objects can work for your requirements and having to worry about types is not entirely necessary. It also means that an object is rarely only one instance of a class meaning it can be dynamically changed at run time. Keeping this polymorphism of objects and variables allows for some unique design patterns and potential implementations that would perhaps not be possible or require more in-depth setup in C#.

This much broader form of polymorphism in python also has a knock effect with things like encapsulation and inheritance. Class attributes not needing to have static types means they can hold different data types, as well as methods and constructors being able to take different data types as inputs.

These are some of the key differences that I found between C# and Python. Overall both languages are heavily Object-Oriented however there are some key differences that impact the design process of the program. Python has a somewhat simpler form of encapsulation with the absence of protection key words and has a much more free flowing broader form of polymorphism that can have a large impact ton the design of classes and methods. Along with this there are some differences between conventions and syntax as well as in relation to translation of source code.

## References

Hacker Noon. (2017). *I Finally Understand Static vs. Dynamic Typing and You Will Too!*. [online] Available at: https://hackernoon.com/i-finally-understand-static-vs-dynamic-typing-and-you-will-too-ad0c2bd0acc7 [Accessed 20 May 2019].

Kb.iu.edu. (2018). *What is the difference between a compiled and an interpreted program?*. [online] Available at: https://kb.iu.edu/d/agsz [Accessed 20 May 2019].

Voegele, J. (n.d.). *Programming Language Comparison*. [online] Jvoegele.com. Available at: http://www.jvoegele.com/software/langcomp.html [Accessed 20 May 2019].

Li, Y. (n.d.). Object-Oriented Design with Python.