

# Other Languages

Jimmy Trac | 101624964 | 2019HS1

## Other Languages

Object Oriented Programming in other languages

The Main Differences

Key Difference: Duck Typing

What is Typing?

What is a Duck?

Summary

---

## Object Oriented Programming in other languages

Though we've only been using C# up until now, many of the concepts used in the creation of C# programs apply to other languages as well, such as Java, C++, and even Ruby. The concepts and the ability to design Object-Oriented (OO) Programs is not just limited to a single language, though oftentimes certain languages implement OO concepts differently.



This mini-report/article will highlight the differences in OO programming between a very strongly typed language, C#, with a dynamically typed language, Python.

## The Main Differences

	C#	Python
	Object Oriented	Procedural, Object Oriented
<b>Interpreted?</b>	Intermediate Language	Interpreted
<b>Readability</b>	Consistent syntax	Very human-readable. <i>expressive</i>

	C#	Python
<b>Platforms</b>	Microsoft .NET	Interpreters on Windows, Linux, MacOS, etc.
<b>Ease of Use</b>	Similar to Java, Integration with .NET Framework	Dynamically typed, requires less knowledge Easy to read Doesn't force OOP

## Key Difference: Duck Typing

Now we have the more boring stuff out of the way, we can talk about the very interesting development that is Duck Typing. There are two words to break down here, let's start with the latter:

### What is Typing?

*In one line, the difference between Static and Dynamically typed is whether or not you had to say so beforehand.*

Typing, with the word *type* referring to object and variable types, refers to how strongly a language enforces that variables must maintain a strict type. Normally this difference occurs in how a language declares variables. Consider the following:

#### C#

```
int a, b, c; // Explicitly declaring variables

a = 1;
b = 3;

c = a + b;
```

#### Python

```
a = 1 # Note how we're not declaring a type here
b = 3

c = a + b
```

This is the difference between statically and dynamically typed languages. Note that this is not the difference between **strongly** and **weakly** typed languages, as those are related but not the same.

To strongly describe it would be to say that the type is associated with either the compile time or run-time. If the language is statically typed, typically the types of variables are checked at runtime and can be used to determine any sources of error that may arise. A good example of this is simply:

```
c = '5' + 2;
```

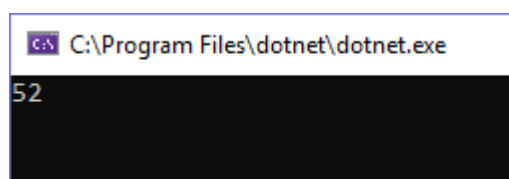
Depending on the language, the operator + can concatenate strings or add numbers. Though the above example may be obvious to human hands, consider Artillery 3's player input. In the game, the players must specify the number of players that are participating--and this is read as a string. If the + operator were to be performed on the string, it would simply concatenate the string. We can test this:

## C#

```
using System;

namespace TestTyping
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("5" + 2);
            Console.ReadLine();
        }
    }
}
```

This gives us an output of:



## Python:

```
# Test for Typing
print("5" + 2)
```

```
Command Prompt

C:\Users\Mikan\Desktop>python -u TestTyping.py
Traceback (most recent call last):
  File "TestTyping.py", line 2, in <module>
    print("5" + 2)
TypeError: can only concatenate str (not "int") to str

C:\Users\Mikan\Desktop>_
```

These examples illustrate the use and dangers of dynamic versus static typing. In our case, Python was able to recognise the issues with concatenating strings and integers, however, the concatenation of strings and integers in C# makes it awfully convenient to display things such as Scores and Health in the form: `Health: 200` or `Score: 500` such as seen in Artillery3.

## What is a Duck?

*"if it walks like a duck and it quacks like a duck, then it must be a duck"*

— *The Duck Test*

The source of incredibly amusing jokes worldwide, the concept of Duck Typing is a genuinely mysterious but well-founded one. Moving on to the first word in the phrase *Duck Typing*, Duck in this case refers to the idea that if an object contained the correct methods without specifically being of a certain type, it can still be treated as that certain type.

Following the classic Duck example, we'll define the characteristics of a Duck as an object that can `swim()` and `layEggs()`, both characteristics of normal ducks. Let's use Python for this example since it'll run.

```
class Duck:
    def swim(self):
        print("Duck is swimming!")
    def layEggs(self):
        print("Duck is laying eggs!")
```

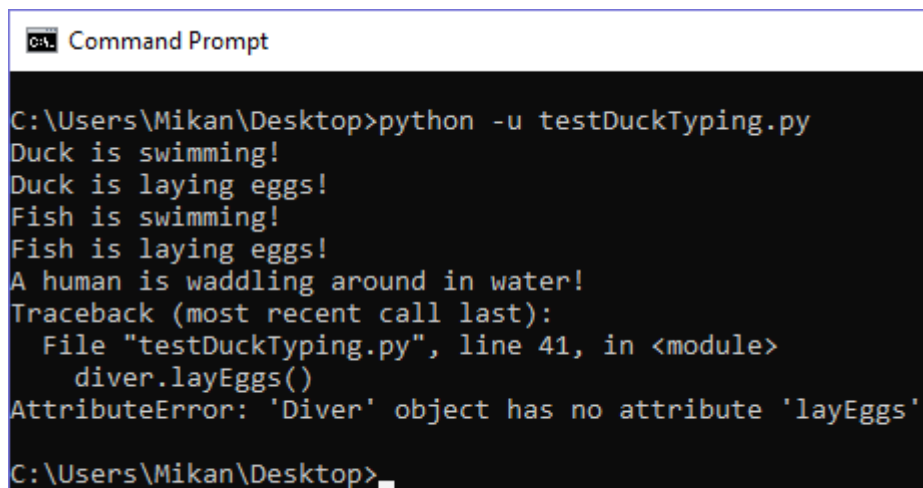
Funnily enough, since we were making an aquatic simulator, let's add some fish and diver:

```
class Fish:
    def swim(self):
        print("Fish is swimming!")
    def layEggs(self):
        print("Fish is laying eggs!")
class Diver:
    def swim(self):
        print("A human is waddling around in water!")
    def saySomethingFunny(self):
        print("MATLAB is a real programming language!")
```

To test our mini-simulator, let's ask some entities to Swim and Lay Eggs:

```
def swim(duck):
    duck.swim()
def layEggs(duck):
    duck.layEggs()
duck = Duck()
fish = Fish()
diver = Diver()
duck.swim()
duck.layEggs()
fish.swim()
fish.layEggs()
diver.swim()
diver.layEggs()
```

Running this code gives us:



```
Command Prompt
C:\Users\Mikan\Desktop>python -u testDuckTyping.py
Duck is swimming!
Duck is laying eggs!
Fish is swimming!
Fish is laying eggs!
A human is waddling around in water!
Traceback (most recent call last):
  File "testDuckTyping.py", line 41, in <module>
    diver.layEggs()
AttributeError: 'Diver' object has no attribute 'layEggs'
C:\Users\Mikan\Desktop>_
```

In essence, duck typing refers to the ability for a language to treat a given object as any other given that it has a suitable method signature.

This is similar to Interfaces in C# whereby classes can inherit method signatures from an interface and be treated as such. In Artillery3, the camera requires a focus point to follow-- this is defined by the `ICameraCanFocus` interface. As long as any object inherits this interface it will be treated as a Focus Point for the camera. Of course, this cannot be done dynamically and especially at runtime without specifying beforehand within C#. On the other hand, Python does not check for object type nor interfaces, but rather, the presence of methods as long as it is supported by the object. If the object is unable to perform such a method, Python throws an error.

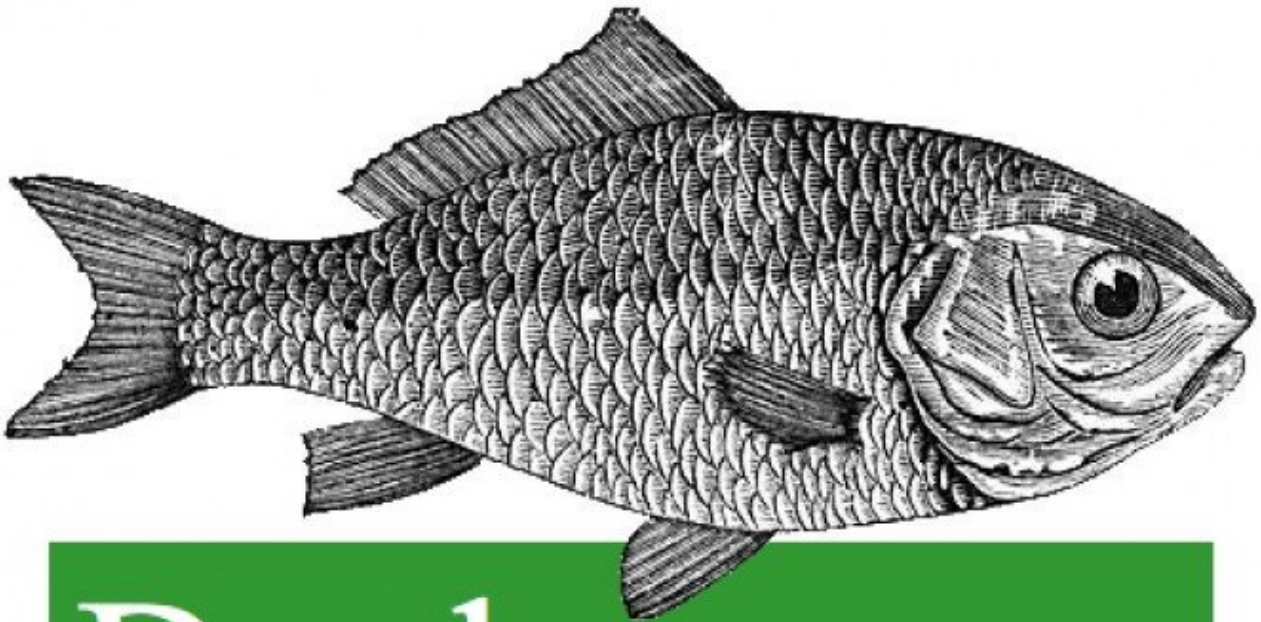
## Summary

Python and C# are almost two sides of the same coin. Python is easy to use and set up, powerful, and very free. On the other hand, using C# is akin to using a company car: everything is taken care of for you, and it feels very professional. The use of both languages depends largely on the application with python generally enjoying more popularity due it's lower bar for entry. Of course, this doesn't make a computer programmer any better, however, a strong understanding of underlying logic is why programmers get paid.

With the key difference between C# and Python discussed in this piece, it highlights the intention of each language: C# is strongly typed as it aims to be consistent. On the other hand, Python aims to be open and expressive, hence, it allows the use of duck typing. These philosophies drive the direction and specialisation of languages, and as a result, leave us with a variety of tools in our programming toolbox to use for just the right situation.

See below for another reason why Duck Typing is a good idea.

*If It Can Swim And Lay Eggs It's A Duck*



# Duck Typing

*Everything Has An Inner Duck*

NOT REILLY

*A Goose*

*Duck Typing: dynamic mechanism allowing to discover that a fish cannot say quack only at runtime ... in production ... on Friday evening, Mario Fusco, Twitter. Last Accessed 12/05/2019.*