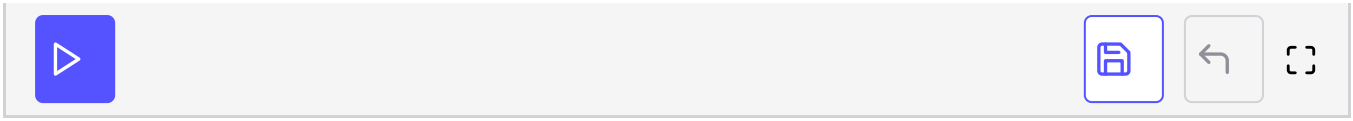
Getting Started

Python 3.5 added an interesting new library called **typing**. This adds type hinting to Python. Type hinting is kind of declaring your functions arguments to have a certain type. However the type hinting is not binding. It's just a hint, so there's nothing preventing the programmer from passing something they shouldn't. This is Python after all. You can read the type hinting specification in PEP 484 or you can just read the theory behind it in PEP 483.

Let's take a look at a simple example:

```
def some_function(number: int, name: str) -> None:
  print("%s entered %s" % (name, number))


print (some_function(13, 'Mike'))
#Mike entered 13
```



This means that **some_function** expects two arguments where the first is an integer and the second is a string. It should also be noted that we have hinted that this function returns None.

Let's back up a bit and write a function the normal way. Then we'll add type hinting to it. In the following example, we have a function that takes list and a name, which in this case would be a string. All it does is check if the name is in the list and returns an appropriate Boolean.

```
def process_data(my_list, name):
    if name in my_list:
      return True
    else:
      return False

if __name__ == '__main__':
    my_list = ['Mike', 'Nick', 'Toby']
```

```
    print( process_data(my_list, 'Mike') )
    print( process_data(my_list, 'John') )
```

Now let's add type hinting to this function:

```
def process_data(my_list: list, name: str) -> bool:
    return name in my_list

if __name__ == '__main__':
    my_list = ['Mike', 'Nick', 'Toby']
    print( process_data(my_list, 'Mike') )
    print( process_data(my_list, 'John') )
```

In this code we hint that the first argument is a list, the second is a string and the return value is a Boolean.

According to PEP 484, "Type hints may be built-in classes (including those defined in standard library or third-party extension modules), abstract base classes, types available in the types module, and user-defined classes". So that means we can create our own class and add a hint.

```
class Fruit:
    def __init__(self, name, color):
      self.name = name
      self.color = color


def salad(fruit_one: Fruit, fruit_two: Fruit) -> list:
    print(fruit_one.name)
    print(fruit_two.name)
    return [fruit_one, fruit_two]

if __name__ == '__main__':
    f = Fruit('orange', 'orange')
    f2 = Fruit('apple', 'red')
    salad(f, f2)
```

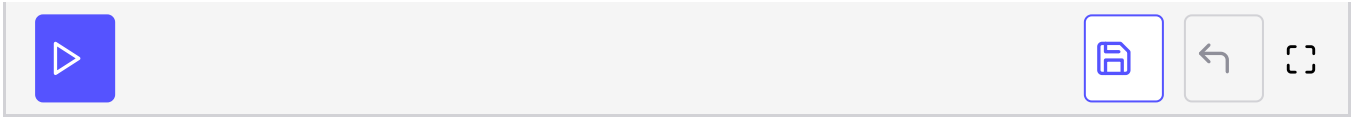
Here we create a simple class and then a function that expects two instances of that class and returns a list object. The other topic that I thought was

interesting is that you can create an Alias. Here's a super simple example:

```
Animal = str

def zoo(animal: Animal, number: int) -> None:
    print("The zoo has %s %s" % (number, animal))

if __name__ == '__main__':
    zoo('Zebras', 10)
```



As you may have guessed, we just aliased the **string** type with the variable **Animal**. Then we added a hint to our function using the Animal alias.