PYTHON CLASS 16

MAKING CLASSES/TYPES

REMEMBER, THERE ARE TWO IMPORTANT THINGS

Information

Actions

A class or type combines both in one place.

EXAMPLE: The Date Class

```
mydate = datetime.date(2022,11,10)
```

Information
(variables)

mydate.day

mydate. month

mydate. year

Actions
(functions)

mydate. today()

mydate. weekday()

mydate. toordinal()

CLASS SYNTAX; HOW TO MAKE A CLASS / TYPE

Most classes have the __init__ function. This is used to setup the variables of the class.

```
class My_Class:
    def __init__(self):
        self.name = ""
        self.number_of_students = 0
```

The class keyword (予約語)

The “self” keyword. Self (自己) means the instance of the class.
We need to use the self keyword to create variables for the class.

CREATE A CLASS INSTANCE (実体;実例)

じったい じつれい

- We use the name of the class to create an instance.

The self argument is always hidden.

```
mc = My_Class()
mc.name = "POP"
mc.number_of_students = 4
```

Remember – the dot allows us to access variables inside the instance of a class

THE IMPORTANCE OF SELF

- See what happens if you leave off self when creating a class.

```
import datetime

class My_Class:
    def __init__(): # check what happens
        name = “”
        number_of_students = 0
        dates = []

c = MyClass() # check what happens
c.name = “POP Programming” # check what happens
c.number_of_students = 3 # check what happens
```

REVIEW

- We have seen this a few times before.

```
my_birthday = datetime.date(1685,3,31)
```

The __init__ function is hidden.

In the example above, when we call datetime.date(year, month, day), this actually calls the __init__ function.

REVIEW

We also see this when we cast -

```
number = input("Enter your favorite number \n")

number = int(number)
```

Creating a new integer from the string.

Behind the scenes, it probably looks like this

```
class int:
    def __init__(self,x):
        if(type(x) == str):
            #do something to change the string
            # to an integer
        else:
            self.value = x

    def __add__(self,other):
        self.value += other
    #more functions...
```

ADDING FUNCTIONS

```
class My_Class:
    def __init__(self):
        self.name = ""
        self.number_of_students = 0
        self.dates = []

    def AddClassDate(self,date):
        self.dates.append(date)

    def RemoveClassDate(self,date):
        self.dates.remove(date)
```

USING FUNCTIONS IN A NEW CLASS / TYPE

```
import datetime

class My_Class:
    def __init__(self):
        self.name = “”
        self.number_of_students = 0
        self.dates = []

    def AddClassDate(self,date):
        self.dates.append(date)
....

c = My_Class()
d = datetime.date(2022,10,3)
c.AddClassDate(d)
c.name = “POP Programming”
c.number_of_students = 3
```

Notice that we never use the “self” argument when we make a new My_Class variable, or when we use a function of My_Class

ANOTHER WAY

```
import datetime

class My_Class:
        def __init__(self):
                self.name = “”
                self.number_of_students = 0
                self.dates = []

        def AddClassDate(self,date):
                self.dates.append(date)
....

c = My_Class()
d = datetime.date(2022,10,3)
c.AddClassDate(d)
c.name = “POP Programming”
c.number_of_students = 3
```

```
import datetime

class My_Class:
        def __init__(self,n,s,d):
                self.name = n
                self.number_of_students = s
                self.dates = d

        def AddClassDate(self,date):
                self.dates.append(date)
....
d = [datetime.date(2022,10,3), datetime.date(2022,10,10)]

c = My_Class(“POP Programming”,3,d)
nd = datetime.date(2022,10,17)
c.AddClassDate(nd)
```

MAKING A NEW CLASS / TYPE: ANOTHER EXAMPLE

```
import datetime

class My_Class:
        def __init__(self,n,s,d):
                self.name = n
                self.number_of_students = s
                self.dates = d

        def AddClassDate(self,date):
                self.dates.append(date)
....
d = [datetime.date(2022,10,3), datetime.date(2022,10,10)]

c = My_Class("POP Programming",3,d)
nd = datetime.date(2022,10,17)
c.AddClassDate(nd)
```

We can pass the information to the new class variable when we make it.

Both ways are OK. Which way is best depends on the program design and the situation.

Functions with or without self

```
import datetime
import random

class My_Class:
    def __init__(self,n,s,d):
        self.name = n
        self.number_of_students = s
        self.dates = d

    def AddClassDate(self,date):
        self.dates.append(date)

    def GetRandomDateInYear():
        year = datetime.date.today().year
        month = random.randint(1,13)
        day =  random.randint(1,29) # there is a way to do this better, but it is complicated
        return  datetime.date(year, month, day)
```

This function doesn't need self, because it does not use any functions or variables from the class.

REMEMBER OUR VENDING MACHINE PROJECT? NOW, LET'S USE LISTS **AND** CLASSES!!!!

ADDING A PRODUCT CLASS TO OUR PYTHON PROJECT

- What variables will be important for a product? (information)
- What functions will the product class need? (actions)

MAKING THE CHANGES

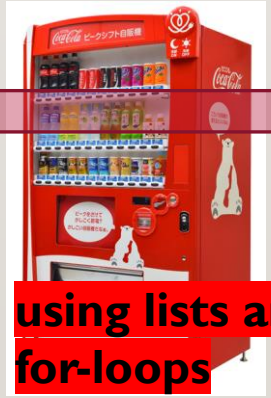
- Go through the Vending Machine Project and update it so that we have a product class, and a list of products.

- Update the code and the functions we created last time so that they will work correctly with a list of products.

ORIGINAL AND NEW VERSION COMPARISON

- Compare the original vending machine project and the new version.
- How does using classes change the code?
- Is this better or worse, overall? Why?

original



MORE VENDING MACHINE IMPROVEMENTS

- Can you think of any other ways to improve the vending machine project?