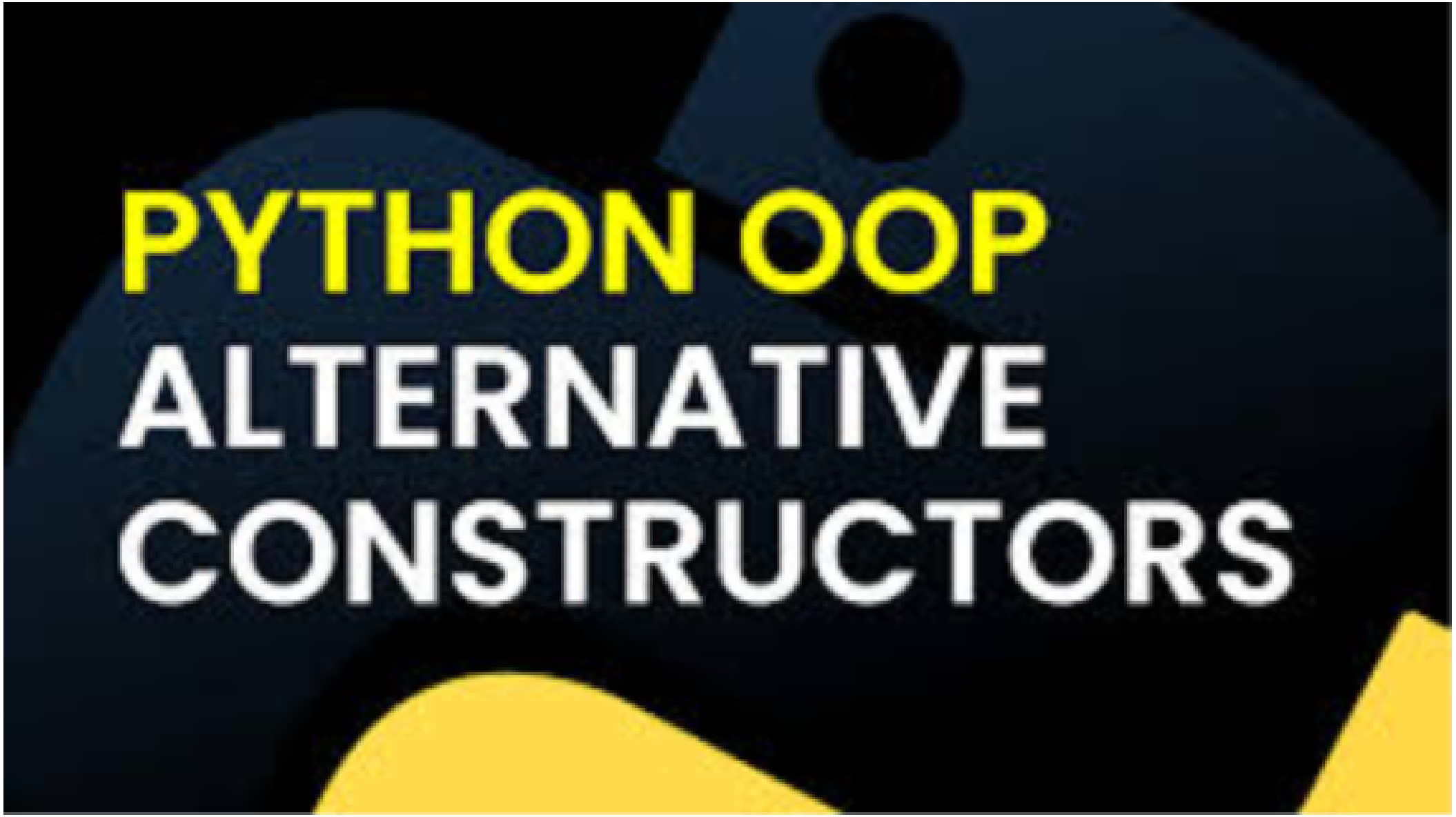
Python


PYTHON OOP
ALTERNATIVE
CONSTRUCTORS

We learned about constructor and its functionality in tutorial #55, where we had to set all the values as parameters to a constructor. Now we will learn how to use a method as a constructor. It has its own advantages. By using a method as a constructor, we would be able to pass the values to it using a string.

Note that we are talking about a class method, not a static method.

The parameters that we have to pass to our constructor would be the class i.e., cls and the string containing the parameters. Moving on towards the working, we have to use a function "**split()**," that will divide the string into parts. And the parts as results will be stored in a list. We can now pass the parameters to the constructor using the index numbers of the list or by the concept of *args (discussed in tutorial#43).

split():

Let us have a brief overview of the split() function. What split() does is, it takes a separator as a parameter. If we do not provide any, then the default separator is any whitespace it encounters. Else we can provide any separator to it such as full stop, hash, dash, colon, etc. After separating the string into parts, the split() function stores it into a list in a sequence. For example:

```
text = "Python tutorial for absolute beginners."
 t = text.split()
 print(t)
```

Here, we are not providing it any separator as a parameter, so it will automatically divide, taking whitespace as a separator.
The output will be a list, such as:
['Python', 'tutorial', 'for', 'absolute', 'beginners.]

Example of Class methods - alternative constructor:

```
class Date:
    def __init__(self, year, month, day):
        self.year = year
        self.month = month
        self.day = day

@classmethod
    def from_dash(cls,string):
          return cls(*string.split("-"))

date1=Date.from_dash("2008-12-5")
print(date1.year)
#Output: 2008
```

If we want multiple and independent "constructors", we can use class methods. They are usually called factory methods. It does not invoke the default constructor __init__.In the above example, we split the string based on the "-" operator. We first create a class method as a constructor that takes the string and split it based on the specified operator. For this purpose, we use a split() function, which takes the separator as a parameter. This alternative constructor approach is useful when we have to deal with files containing string data separated by a separator.

new*

```
1  class employee:
2      def __init__(self, name, salary):
3          self.name=name
4          self.salary=salary
5      @classmethod
6      def fromstr(cls, string):
7          return cls(string.split("-")[0], string.split("-")[1])
8  e=employee("Ajay", 99999)
9  print(e.name)
10 print(e.salary)
11 string="Ajay-20000"
12 e1=employee.fromstr(string)
13 print(e1.name)
14 print(e1.salary)
15
```


TAB
Ajay
99999
Ajay
20000
[Program finished]

```
21 class person:
22     def __init__(self, name, age):
23         self.name=name
24         self.age=age
25     @classmethod
26     def from_string(cls, string):
27         name, age=string.split(",")
28         return cls(name, int(age))
29
30 string="Ajay,17"
31 person=person.from_string(string)
32 print(person.name)
33 print(person.age)
34 print(type(person.name))
35 print(type(person.age))
36
```


TAB
Ajay
17
<class 'str'>
<class 'int'>
[Program finished]