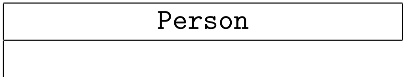
Designing and Creating Classes in Python

Designing and Creating Classes in Python

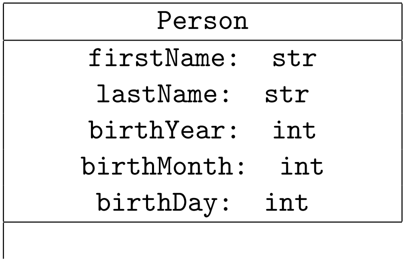
UML Diagrams and Python Class Syntax

Review of OOP

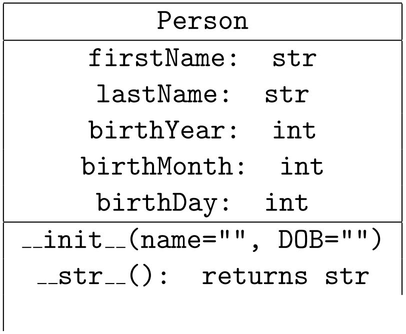
Review of OOP



Review of OOP

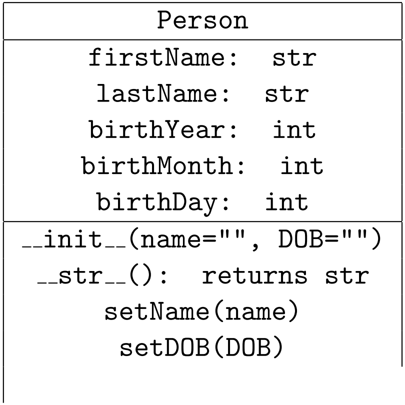


Review of OOP



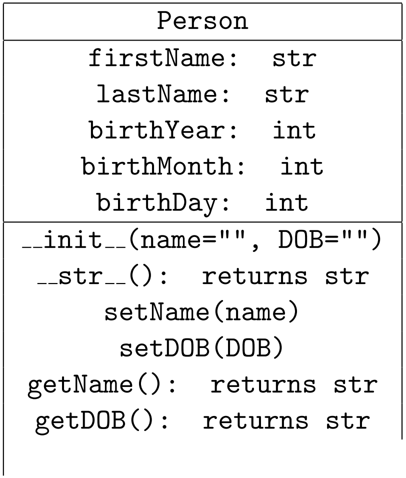
```
Person
firstName: str
lastName: str
birthYear: int
birthMonth: int
birthDay: int
__init__(name="", DOB="")
__str__(): returns str
```

Review of OOP



```
Person
firstName: str
lastName: str
birthYear: int
birthMonth: int
birthDay: int
__init__(name="", DOB="")
__str__(): returns str
setName(name)
setDOB(DOB)
```

Review of OOP



Person
firstName: str
lastName: str
birthYear: int
birthMonth: int
birthDay: int
`__init__(name="", DOB="")`
`__str__()`: returns str
setName(name)
setDOB(DOB)
getName(): returns str
getDOB(): returns str

Review of OOP

Person
firstName: str
lastName: str
birthYear: int
birthMonth: int
birthDay: int
__init__(name="", DOB="")
__str__(): returns str
setName(name)
setDOB(DOB)
getName(): returns str
getDOB(): returns str
canVote(): returns bool

Review of OOP

Person
firstName: str
lastName: str
birthYear: int
birthMonth: int
birthDay: int
__init__(name="", DOB="")
__str__(): returns str
setName(name)
setDOB(DOB)
getName(): returns str
getDOB(): returns str
canVote(): returns bool

```
john = Person()
```

Review of OOP

Person
firstName: str
lastName: str
birthYear: int
birthMonth: int
birthDay: int
__init__(name="", DOB="")
__str__(): returns str
setName(name)
setDOB(DOB)
getName(): returns str
getDOB(): returns str
canVote(): returns bool

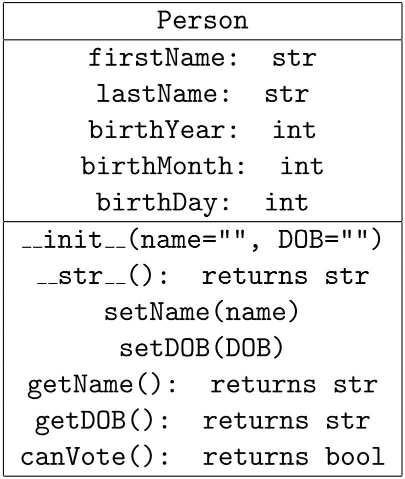
```
john = Person()
john.setName("John
Johnson")
```

Review of OOP

Person
firstName: str
lastName: str
birthYear: int
birthMonth: int
birthDay: int
__init__(name="", DOB="")
__str__(): returns str
setName(name)
setDOB(DOB)
getName(): returns str
getDOB(): returns str
canVote(): returns bool

```
john = Person()
john.setName("John
Johnson")
john.setDOB("01/01/2001")
```

Review of OOP



Person
firstName: str
lastName: str
birthYear: int
birthMonth: int
birthDay: int
__init__(name="", DOB="")
__str__(): returns str
setName(name)
setDOB(DOB)
getName(): returns str
getDOB(): returns str
canVote(): returns bool

```
john = Person()
john.setName("John
Johnson")
john.setDOB("01/01/2001")
print(john)
```

Review of OOP

Person
firstName: str lastName: str birthYear: int birthMonth: int birthDay: int
__init__(name="", DOB="") __str__(): returns str setName(name) setDOB(DOB) getName(): returns str getDOB(): returns str canVote(): returns bool

```
john = Person()
john.setName("John
Johnson")
john.setDOB("01/01/2001")
print(john)
if john.canVote():
    print(john.getName() +
              " can vote")
```

UML Diagrams

Person
firstName: str
lastName: str
birthYear: int
birthMonth: int
birthDay: int
__init__(name="", DOB="")
__str__(): returns str
setName(name)
setDOB(DOB)
getName(): returns str
getDOB(): returns str
canVote(): returns bool

UML Diagrams

Person
firstName: str
lastName: str
birthYear: int
birthMonth: int
birthDay: int
__init__(name="", DOB="")
__str__(): returns str
setName(name)
setDOB(DOB)
getName(): returns str
getDOB(): returns str
canVote(): returns bool

`https://yuml.me/`

Creating Classes

Creating Classes

```
class Person():
```

Creating Classes

```
class Person():
    """ Docstring for Person here """
```

Creating Classes

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
```

Creating Classes

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
```

Creating Classes

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
        self.firstName = ""
```

Creating Classes

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
        self.firstName = ""
        self.lastName = ""
```

Creating Classes

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
        self.firstName = ""
        self.lastName = ""
        self.birthMonth = 0
```

Creating Classes

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
        self.firstName = ""
        self.lastName = ""
        self.birthMonth = 0
        self.birthDay = 0
```

Creating Classes

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
        self.firstName = ""
        self.lastName = ""
        self.birthMonth = 0
        self.birthDay = 0
        self.birthYear = 0
```

Creating Classes

```
class Person():
    """ Docstring for Person here """
    def __init__(self):
        """ Docstring for constructor here """
        self.firstName = ""
        self.lastName = ""
        self.birthMonth = 0
        self.birthDay = 0
        self.birthYear = 0
        return
```

Creating Classes (continued)

Creating Classes (continued)

```
class Person():
```

Creating Classes (continued)

```
class Person():
     ⋮
```

Creating Classes (continued)

```
class Person():
    ⋮
    def __str__(self):
```

Creating Classes (continued)

```
class Person():
    ⋮
    def __str__(self):
        """ Docstring for __str__ here """
```

Creating Classes (continued)

```
class Person():
      ⋮
    def __str__(self):
        """ Docstring for __str__ here """
        return f"{self.firstName} ..."
```

Creating Classes (continued)

Creating Classes (continued)

```
class Person():
```

Creating Classes (continued)

```
class Person():
    ⋮
```

Creating Classes (continued)

```
class Person():
        ⋮
    def setName(self, name):
```

Creating Classes (continued)

```
class Person():
      ⋮
    def setName(self, name):
        """ Docstring for setName here """
```

Creating Classes (continued)

```
class Person():
         ⋮
    def setName(self, name):
        """ Docstring for setName here """
        self.firstName = name.split()[0]
```

Creating Classes (continued)

```
class Person():
        ⋮
    def setName(self, name):
        """ Docstring for setName here """
        self.firstName = name.split()[0]
        self.lastName = name.split()[1]
```

Creating Classes (continued)

```
class Person():
         ⋮
    def setName(self, name):
        """ Docstring for setName here """
        self.firstName = name.split()[0]
        self.lastName = name.split()[1]
        return
```

Creating Classes (continued)

Creating Classes (continued)

```
class Person():
```

Creating Classes (continued)

```
class Person():
        ⋮
```

Creating Classes (continued)

```
class Person():
    ⋮
    def getName(self):
```

Creating Classes (continued)

```
class Person():
    ⋮
    def getName(self):
        """ Docstring for getName here """
```

Creating Classes (continued)

```
class Person():
       ⋮
    def getName(self):
        """ Docstring for getName here """
        return f"{self.firstName} {self.lastName}"
```

Creating Classes (continued)

Creating Classes (continued)

```
class Person():
```

Creating Classes (continued)

```
class Person():
      ⋮
```

Creating Classes (continued)

```
class Person():
    ⋮
    def canVote(self):
```

Creating Classes (continued)

```
class Person():
      ⋮
    def canVote(self):
        """ Docstring for canVote here """
```

Creating Classes (continued)

```
class Person():
     ⋮
    def canVote(self):
        """ Docstring for canVote here """
        from dataetime import datetime
```

Creating Classes (continued)

```
class Person():
      ⋮
    def canVote(self):
        """ Docstring for canVote here """
        from dataetime import datetime
        todayDay = datetime.now().day
        todayMonth = datetime.now().month
        todayYear = datetime.now().year
```

Creating Classes (continued)

```
class Person():
      ⋮
    def canVote(self):
        """ Docstring for canVote here """
        from dataetime import datetime
        todayDay = datetime.now().day
        todayMonth = datetime.now().month
        todayYear = datetime.now().year
        if todayYear > self.birthYear + 18:
```

Creating Classes (continued)

```
class Person():
      ⋮
    def canVote(self):
        """ Docstring for canVote here """
        from dataetime import datetime
        todayDay = datetime.now().day
        todayMonth = datetime.now().month
        todayYear = datetime.now().year
        if todayYear > self.birthYear + 18:
            return True
        elif ...:
            ...
```

Class modules

Class modules

The `Person` class is saved in the `person.py` file.

Class modules

The `Person` class is saved in the `person.py` file.

So to use:

Class modules

The `Person` class is saved in the `person.py` file.

So to use:

```
import person
```

Class modules

The `Person` class is saved in the `person.py` file.

So to use:

```
import person
tyler = person.Person()
```

Concept Check!

What is wrong with the following class definition?

```
class Dog():
    def __init__(name, breed):
        self.name = name
        self.breed = breed
        return
    def isGoodBoy():
        return True
```

Concept Check!

What is wrong with the following class definition?

```
class Dog():
    def __init__(name, breed):
        self.name = name
        self.breed = breed
        return
    def isGoodBoy():
        return True
```

✗ **Missing `self` as input for both methods**

Solution

```
class Dog():
    def __init__(self, name, breed):
        self.name = name
        self.breed = breed
        return
    def isGoodBoy(self):
        return True
```