Introduction to Objects

Introduction to Objects

Working with Object Instances

Review of Procedural Programming

Review of Procedural Programming

```
def func1(x, y):
    statement1
    statement2
        ⋮
    return z

def func2():
    statement50
    statement51
        ⋮
    return

    ⋮
```

Review of Procedural Programming

```
def func1(x, y):
    statement1
    statement2
        ⋮
    return z

def func2():
    statement50
    statement51
        ⋮
    return

    ⋮
```

```
a few input statements
out1 = func1(input1,
input2)
func2()
out2 = func3(out1)
    ⋮
```

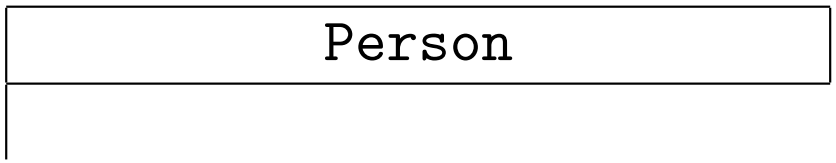
Review of Procedural Programming

```
def func1(x, y):
    statement1
    statement2
        ⋮
    return z

def func2():
    statement50
    statement51
        ⋮
    return

    ⋮
```

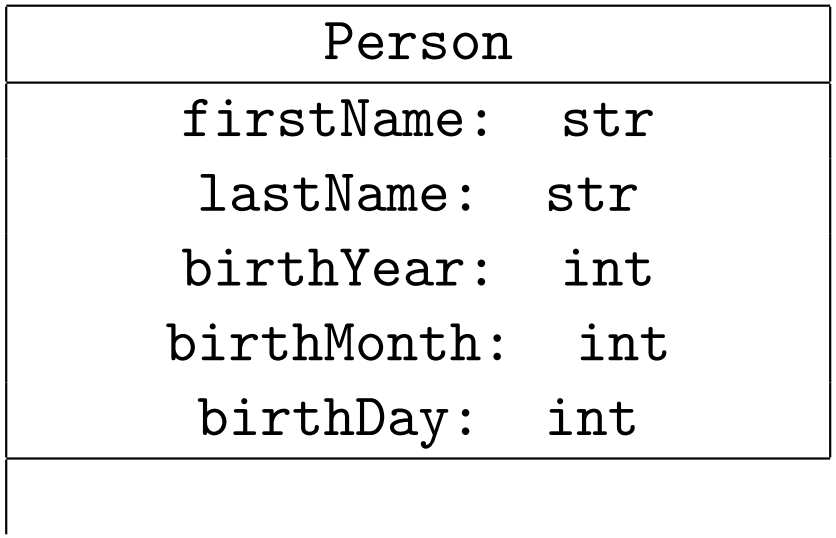
```
a few input statements
out1 = func1(input1,
input2)
func2()
out2 = func3(out1)
    ⋮
print(func10(out9, out10))
```

Object-Oriented Programming

Object-Oriented Programming



Object-Oriented Programming



Object-Oriented Programming

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

Object-Oriented Programming

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

Object-Oriented Programming

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

Object-Oriented Programming

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

Object-Oriented Programming

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()
```

Object-Oriented Programming

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")
```

Object-Oriented Programming

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")
```

Object-Oriented Programming

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)
```

Object-Oriented Programming

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

Encapsulation

Encapsulation

- Combine data and functions together in a single “object”

Encapsulation

- Combine data and functions together in a single “object”
- Person class had data:

Encapsulation

- Combine data and functions together in a single “object”
- Person class had data:
  - `firstName`

Encapsulation

- Combine data and functions together in a single “object”
- Person class had data:
  - `firstName`
  - `lastName`

Encapsulation

- Combine data and functions together in a single “object”
- Person class had data:
  - `firstName`
  - `lastName`
  - `birthYear`

Encapsulation

- Combine data and functions together in a single “object”
- Person class had data:
  - `firstName`
  - `lastName`
  - `birthYear`
  - ...

Encapsulation

- Combine data and functions together in a single “object”
- Person class had data:
  - `firstName`
  - `lastName`
  - `birthYear`
  - ...
- and *methods*:

Encapsulation

- Combine data and functions together in a single "object"
- Person class had data:
  - `firstName`
  - `lastName`
  - `birthYear`
  - ...
- and *methods*:
  - `setName(name)`

Encapsulation

- Combine data and functions together in a single "object"
- Person class had data:
  - `firstName`
  - `lastName`
  - `birthYear`
  - ...
- and *methods*:
  - `setName(name)`
  - `getName()`

Encapsulation

- Combine data and functions together in a single "object"
- Person class had data:
  - `firstName`
  - `lastName`
  - `birthYear`
  - ...
- and *methods*:
  - `setName(name)`
  - `getName()`
  - `__init__()`

Encapsulation

- Combine data and functions together in a single "object"
- Person class had data:
  - `firstName`
  - `lastName`
  - `birthYear`
  - ...
- and *methods*:
  - `setName(name)`
  - `getName()`
  - `__init__()` – *constructor*

Terminology

Terminology

- `Person`

Terminology

- `Person` `class`

Terminology

- `Person` `class`
- `john`

Terminology

- `Person  class`
- `john  instance of Person`

Terminology

- `Person` `class`
- `john` `instance of Person` `Person object`

Terminology

- `Person` `class`
- `john` `instance of Person` `Person object`
- `john.getName()`

Terminology

- `Person` `class`
- `john` `instance of Person` `Person object`
- `john.getName()` *method* of the `Person` class

Terminology

- `Person` `class`
- `john` `instance of Person` `Person object`
- `john.getName()` *method* of the `Person` class
- `john.firstName`

Terminology

- `Person` `class`
- `john` `instance of Person` `Person object`
- `john.getName()` *method* of the `Person` class
- `john.firstName` *attribute* of the `Person` class

Familiar built-in classes

Familiar built-in classes

- `list`

Familiar built-in classes

- `list`
- `tuple`

Familiar built-in classes

- `list`
- `tuple`
- `str`

Familiar built-in classes

- `list`
- `tuple`
- `str`
- `dict`

Familiar built-in classes

- `list`
- `tuple`
- `str`
- `dict`
- `set`

Concept Check!

Perform the following tasks, given the Car class UML diagram:

Car
make: str model: str year: int
__init__(make, model, year) setInfo(make, model, year) getInfo(): returns make, model, year

1. Create a car object
2. Get the car's info

Concept Check!

Perform the following tasks, given the Car class UML diagram:

Car
make: str model: str year: int
__init__(make, model, year) setInfo(make, model, year) getInfo(): returns make, model, year

1. Create a car object `myCar = Car("Toyota", "Corolla", 2015)`
2. Get the car's info

Concept Check!

Perform the following tasks, given the Car class UML diagram:

Car
make: str model: str year: int
__init__(make, model, year) setInfo(make, model, year) getInfo(): returns make, model, year

1. Create a car object
```
myCar = Car("Toyota", "Corolla", 2015)
```
2. Get the car's info
```
make, model, year = myCar.getInfo()
```

OOP vs. Procedural Programming

OOP vs. Procedural Programming

Each has pros and cons

OOP vs. Procedural Programming

Each has pros and cons

"Best technique" is situational

OOP vs. Procedural Programming

Each has pros and cons

"Best technique" is situational

Rule of thumb:

OOP vs. Procedural Programming

Each has pros and cons

"Best technique" is situational

Rule of thumb:

- Use procedural programming when offering one-time services (like converting a single temperatures from Fahrenheit to Celsius)

OOP vs. Procedural Programming

Each has pros and cons

"Best technique" is situational

Rule of thumb:

- Use procedural programming when offering one-time services (like converting a single temperatures from Fahrenheit to Celsius)
- Use OOP when you are providing a service that is driven by data, such as implementing a database

OOP vs. Procedural Programming

Each has pros and cons

"Best technique" is situational

Rule of thumb:

- Use procedural programming when offering one-time services (like converting a single temperatures from Fahrenheit to Celsius)
- Use OOP when you are providing a service that is driven by data, such as implementing a database
- No one says these have to be mutually exclusive, you can (and should) mix both in a single project