

# Classes

**David H Smith IV**

**University of Illinois Urbana-Champaign**

**Thur, Dec 02 2021**

# Reminders

# Reminders

- Usual mix of homework (see the website)
- Quiz next Thursday, practice quiz will be up Saturday
- Practice final will be up next Tuesday.

## Review + New Stuff: Class Constructors

# Poll Question: Constructors

Which of the following are valid constructor function defs?

- ❶ `def __init__(self):`
- ❷ `def __init__(self, foo):`
- ❸ `def _init(self):`
- ❹ `def __init__(self, foo, *args, **kwargs):`
- ❺ `def __init__(self, foo, bar=0, *baz, **qux):`
- ❻ `def __init__(self, bar=0, foo, **qux, *baz):`

**Consider each individually in groups and I will go through and ask which are valid.**

# Poll Question: Setting Things Up

What is printed in when the following code is run?

```
class Person:
    def __init__(self, name, age):
        myname = name
        myage = age
    def get_name(self):
        print(myname)

p1 = Person("Dave", 22)
print(p1.get_name())
```

- ☐ A NameError
- ☐ B 22
- ☐ C Dave
- ☐ D self

## Poll Question: Setting Things Up

What is printed in when the following code is run?

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def get_date_of_birth(current_year):
        return current_year - self.age

p1 = Person("Dave", 22)
print(p1.get_date_of_birth(2021))
```

- ☐ A 1999
- ☐ B -1977
- ☐ C NameError
- ☐ D TypeError

# Poll Question: Calling Class Functions in the Class

What code should replace the question marks?

```
class People:
    def __init__(self, *people):
        self.people_list = people

    def _get_age_difference(self, p1, p2):
        return abs(p1.age - p2.age)

    def get_age_differences(self, person, current_year):
        for otherperson in self.people_list:
            if otherperson is not person:
                n1 = otherperson.name
                n2 = person.name
                diff = ??
                print(n1, n2, diff)
```

- ☐ A self.\_get\_age\_difference(otherperson, person)
- ☐ B \_get\_age\_difference(otherperson, person)
- ☐ C \_get\_age\_difference(self, otherperson, person)



# Class Interfaces, Internal Methods, and Encapsulation

# Encapsulation: Private vs Public

- Ⓐ Encapsulation is a fundamental concept in object oriented programming (OOP).

# Encapsulation: Private vs Public

- Ⓐ Encapsulation is a fundamental concept in object oriented programming (OOP).
- Ⓑ Private vs Public Variables:

# Encapsulation: Private vs Public

- Ⓐ Encapsulation is a fundamental concept in object oriented programming (OOP).
- Ⓑ Private vs Public Variables:
  - Ⓐ **Public:** The values/methods can be used outside of the class.

# Encapsulation: Private vs Public

- Ⓐ Encapsulation is a fundamental concept in object oriented programming (OOP).
- Ⓑ Private vs Public Variables:
  - Ⓐ **Public:** The values/methods can be used outside of the class.
  - Ⓑ **Private** The values/methods can't be used outside of the class.

# Encapsulation: Private vs Public

- Ⓐ Encapsulation is a fundamental concept in object oriented programming (OOP).
- Ⓑ Private vs Public Variables:
  - Ⓐ **Public:** The values/methods can be used outside of the class.
  - Ⓑ **Private** The values/methods can't be used outside of the class.
- Ⓒ **Python doesn't have encapsulation!** (I have opinions on this...)

```
def Foo:
    _private_func(self):
        #...

    public_func(self):
        self._private_func() #Good practice
        #...

f = Foo()
f._private_func() #Not good practice
f.pubic_func() #Good practice
```

```
def Foo:
    _private_func(self):
        #...

    public_func(self):
        self._private_func() #Good practice
        #...

f = Foo()
f._private_func() #Not good practice
f.pubic_func() #Good practice
```

- Ⓐ Functions/variables *intended* to only be used in the classes start with an "\_". Python's version of private.



```
def Foo:
    _private_func(self):
        #...

    public_func(self):
        self._private_func() #Good practice
        #...

f = Foo()
f._private_func() #Not good practice
f.pubic_func() #Good practice
```

- Ⓐ Functions/variables *intended* to only be used in the classes start with an "\_". Python's version of private.
- Ⓑ Functions/variables *intended* to be used anywhere don't start with an "\_".

```
def Foo:
    _private_func(self):
        #...

    public_func(self):
        self._private_func() #Good practice
        #...

f = Foo()
f._private_func() #Not good practice
f.pubic_func() #Good practice
```

- Ⓐ Functions/variables *intended* to only be used in the classes start with an "\_". Python's version of private.
- Ⓑ Functions/variables *intended* to be used anywhere don't start with an "\_".
- Ⓒ **This is only by convention and has no impact on how the code runs or is interpreted.**

# Overloading

# Common Operators to Overload

# Example

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __sub__(self, other):
        return self.age - other.age
```

```
p1 = Person("Alice", 22)
p2 = Person("Bob", 27)
age_difference = p2 - p1
```

- > → `__gt__(self, other)`
- >= → `__ge__(self, other)`
- < → `__lt__(self, other)`
- <= → `__le__(self, other)`
- == → `__eq__(self, other)`

- - → `__sub__(self, other)`
- + → `__add__(self, other)`
- \* → `__mul__(self, other)`
- / → `__truediv__(self, other)`
- % → `__mod__(self, other)`
- → `__str__(self)`

# Iteration and Subscripting Overloads

To define the behaviour of the class when (1) a collection are being iterated over or a (2) single instance is being unpacked use the `__iter__(self)` function.

```
class Foo:
    def __init__(self, end):
        self.end = end

    def __iter__(self):
        """Commonly uses a generator"""
        c = 0
        while c < self.end:
            yield c
            c += 1

x = Foo(10)
for i in foo:
    print(i)
```



# Subscripting Overloads

To define the behaviour of the `[]` operator when using variable unpacking or iteration use the `__getitem__(self)` function.

```
class Foo:
    def __init__(self, start, end):
        self.start = start
        self.end = end

    def __getitem__(self, item):
        """Commonly uses a generator"""
        value = self.start + item
        if value < self.end:
            return value

x = Foo(2,10)
print(x[3])
```