

```
class myclass:
    x = 10
    @classmethod
    def f1(cls): # class method or function
        print('myclass:f1( )')
        print('myclass.x =',cls.x)
    def f2(self):
        print('myclass instance method:f2( )')

class yourclass:
    x = 20
    @classmethod
    def f1(cls): # class method or function
        print('yourclass:f1( )')
        print('yourclass.x =',cls.x)
    def f2(self):
        print('yourclass instance method:f2( )')
```

```
class myclass:
    x = 10
    @classmethod
    def f1(cls): # class method or function
        print('myclass class method:f1( )')
        print('myclass.x =',cls.x)
    def f2(self):
        print('myclass instance method:f2( )')
    @staticmethod
    def f3():
        print('myclass static method:f3( )')
```

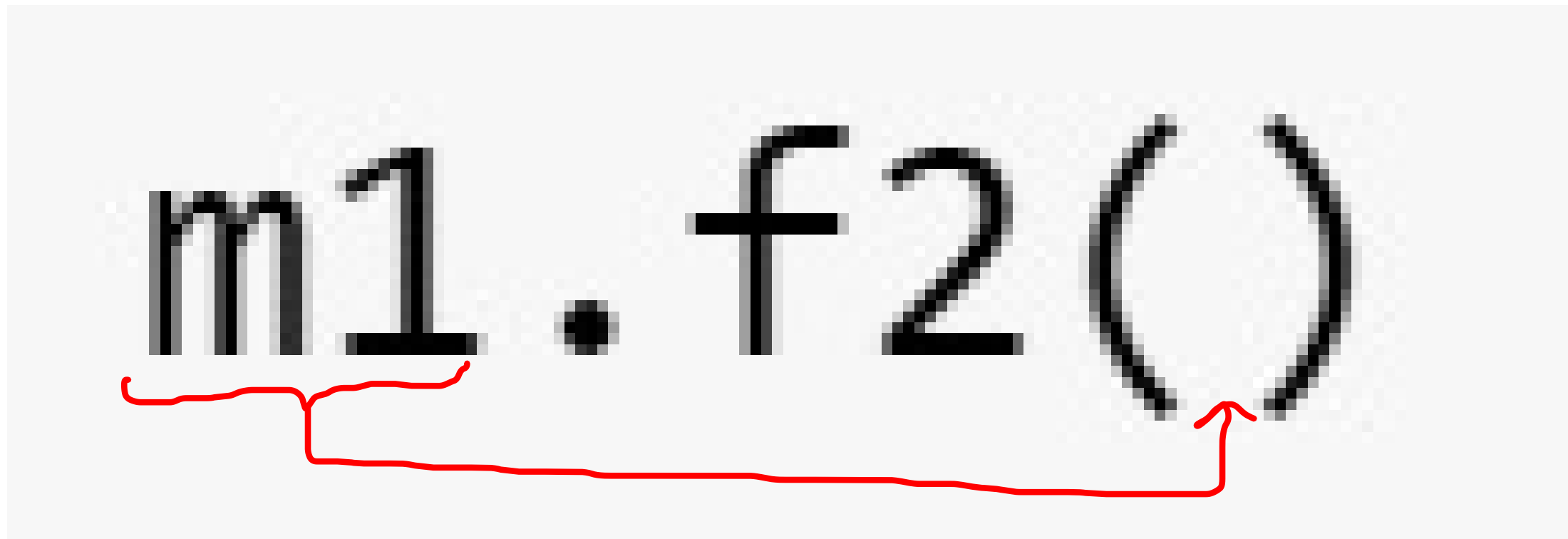
class method

```
myclass.f1()
```



by default class name implicitly supplied as a
first argument

object method or instance method



The diagram shows the code `m1.f2()` in a monospace font. A red bracket is drawn under the `m1` and the dot, extending to the opening parenthesis of the function call. This indicates that the object reference `m1` is implicitly supplied as the first argument to the method `f2`.

by default object reference implicitly supplied as a
first argument

Static method

```
myclass.f3()
```

* implicitly default arguments will not be supplied

class method

```
myclass.f1()
```

```
myclass class method:f1( )  
myclass.x = 10
```

```
m1.f1()
```

```
myclass class method:f1( )  
myclass.x = 10
```

static method

```
myclass.f3()
```

```
myclass static method:f3( )
```

```
m1.f3()
```

```
myclass static method:f3( )
```