Lecture 7: More functions and classes

Inheritance and lambdas

Functions as arguments

```
1   BASE_VAL = 100
2   #https://www.python.org/dev/peps/pep-0008/#constants
3
4   def value_tester(a, b):
5       val = a + b
6       if val > BASE_VAL:
7           statement = 'bigger than'
8       elif val == BASE_VAL:
9           statement = 'the same as'
10      else:
11          statement = 'smaller than'
12
13      print(f'The value is {statement} the base value.')
```

Functions as arguments

```python
BASE_VAL = 100
#https://www.python.org/dev/peps/pep-0008/#constants

def value_tester(a, b):
    val = a + b
    if val > BASE_VAL:
        statement = 'bigger than'
    elif val == BASE_VAL:
        statement = 'the same as'
    else:
        statement = 'smaller than'

    print(f'The value is {statement} the base value.')
```

```
In [2]: value_tester(10, 10)
The value is smaller than the base value.
```

Functions as arguments

```
19  def value_tester(a, b):
20      val = sum([a, b])
21      if val > BASE_VAL:
22          statement = 'bigger than'
23      elif val == BASE_VAL:
24          statement = 'the same as'
25      else:
26          statement = 'smaller than'
27
28      print(f'The value is {statement} the base value.')
```

```
In [10]: value_tester(10, 10)
The value is smaller than the base value.
```

Interlude: unpacking notation 2

```
In [11]: x, y = [10, 20]
    ...: print(x)
    ...: print(y)
10
20
```

From lecture 5

2 variables, 3 values

```
In [12]: x, y = [10, 20, 30]
Traceback (most recent call last):

  File "<ipython-input-12-be2cdddd258d>", line 1, in <module>
    x, y = [10, 20, 30]

ValueError: too many values to unpack (expected 2)
```

Interlude: unpacking notation 2

2 variables, 3 values

```
In [12]: x, y = [10, 20, 30]
Traceback (most recent call last):

  File "<ipython-input-12-be2cdddd258d>", line 1, in <module>
    x, y = [10, 20, 30]

ValueError: too many values to unpack (expected 2)
```

```
In [15]: x, *y = [10, 20, 30]
    ...: print(x)
    ...: print(y)
10
[20, 30]
```

Using unpacking notation with arguments

```
44  def value_tester(*ab):
45      val = sum(ab)
46      if val > BASE_VAL:
47          statement = 'bigger than'
48      elif val == BASE_VAL:
49          statement = 'the same as'
50      else:
51          statement = 'smaller than'
52
53      print(f'The value is {statement} the base value.')
```

```
In [19]: value_tester(10, 10)
The value is smaller than the base value.
```

Using unpacking notation with arguments

```
44  def value_tester(*ab):
45      val = sum(ab)
46      if val > BASE_VAL:
47          statement = 'bigger than'
48      elif val == BASE_VAL:
49          statement = 'the same as'
50      else:
51          statement = 'smaller than'
52
53      print(f'The value is {statement} the base value.')
```

```
In [19]: value_tester(10, 10)
The value is smaller than the base value.
```

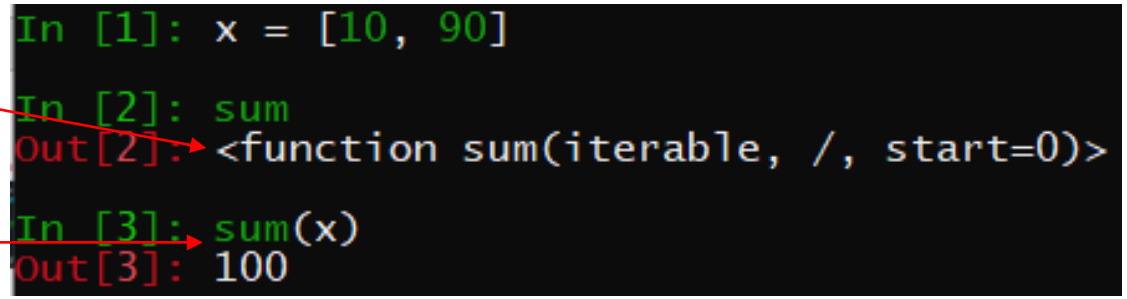
```
In [23]: value_tester(5, 5, 5, 5)
The value is smaller than the base value.

In [24]: value_tester(1)
The value is smaller than the base value.
```

Functions themselves are objects

Function object

Calling the function
to get the result



Functions themselves are objects

Function object

Calling the function
to get the result

```
In [1]: x = [10, 90]

In [2]: sum
Out[2]: <function sum(iterable, /, start=0)>

In [3]: sum(x)
Out[3]: 100
```

```
In [25]: my_func = sum

In [26]: my_func(x)
Out[26]: 100
```

Functions as arguments

```
58  def value_tester(*ab, func=sum):
59      val = func(ab)
60      if val > BASE_VAL:
61          statement = 'bigger than'
62      elif val == BASE_VAL:
63          statement = 'the same as'
64      else:
65          statement = 'smaller than'
66
67      print(f'The value is {statement} the base value.')
```

```
In [21]: value_tester(10, 10)
The value is smaller than the base value.
```

Functions as arguments

```
72  def value_tester(*ab, func=sum):
73      val = func(ab)
74      if val > BASE_VAL:
75          statement = 'bigger than'
76      elif val == BASE_VAL:
77          statement = 'the same as'
78      else:
79          statement = 'smaller than'
80
81      print(f'The value is {statement} the base value.')
```

```
In [5]: import numpy as np
   ...: value_tester(10, 10, func=np.prod)
The value is the same as the base value.
```

Anonymous functions

```
def my_func(a, b):
    return (a + b) / 2


lambda a, b: (a + b) / 2
```

Anonymous functions

```
def my_func(a, b):
    return (a + b) / 2


lambda a, b: (a + b) / 2
```

Anonymous functions

```
def my_func(a, b):
    return (a + b) / 2


lambda a, b: (a + b) / 2
```

Anonymous functions

??

```
def my_func(a, b):
    return (a + b) / 2


lambda a, b: (a + b) / 2
```

Lambda functions

```
In [7]: value_tester(10, 10, func=lambda ab: sum(ab) / len(ab))
The value is smaller than the base value.
```

```
In [9]: value_tester(150, 50, func=lambda ab: sum(ab) / len(ab))
The value is the same as the base value.
```

Lambda functions

```
In [7]: value_tester(10, 10, func=lambda ab: sum(ab) / len(ab))
The value is smaller than the base value.

In [9]: value_tester(150, 50, func=lambda ab: sum(ab) / len(ab))
The value is the same as the base value.
```

arg

body

Lambda functions are:

- Anonymous (don’t have a name)
- One line

Lambda functions

```
In [7]: value_tester(10, 10, func=lambda ab: sum(ab) / len(ab))
The value is smaller than the base value.
```

```
In [9]: value_tester(150, 50, func=lambda ab: sum(ab) / len(ab))
The value is the same as the base value.
```

```
In [14]: def my_func(arg):
    ...:     return arg + 10
    ...: my_func(10)
Out[14]: 20
```

```
In [15]: my_func = lambda arg: arg + 10
    ...: my_func(10)
Out[15]: 20
```

Lambda functions

```
In [7]: value_tester(10, 10, func=lambda ab: sum(ab) / len(ab))
The value is smaller than the base value.
```

```
In [9]: value_tester(150, 50, func=lambda ab: sum(ab) / len(ab))
The value is the same as the base value.
```

```
In [14]: def my_func(arg):
    ...:     return arg + 10
    ...: my_func(10)
Out[14]: 20
```

```
In [15]: my_func = lambda arg: arg + 10
    ...: my_func(10)
Out[15]: 20
```

Equivalent

Wrong!

https://www.python.org/dev/peps/pep-0008/#id51

Class inheritance

```
 99  class Vehicle():
100      def __init__(self, kind):
101          self.kind = kind
102
103      def what_am_i(self):
104          print(f'I am a {self.kind}.')
105
106      def fuel(self):
107          print('I use oil for fuel.')
```

```
In [26]: vehicle = Vehicle('car')
    ...: vehicle.what_am_i()
    ...: vehicle.fuel()
I am a car.
I use oil for fuel.
```

Class inheritance

```
 99  class Vehicle():
100      def __init__(self, kind):
101          self.kind = kind
102
103      def what_am_i(self):
104          print(f'I am a {self.kind}.')
105
106      def fuel(self):
107          print('I use oil for fuel.')
```

```
113  class Bicycle(Vehicle):
114      def fuel(self):
115          print('I am people-powered.')
116
117      def peddle_me(self):
118          print('You have to peddle fast.')
```

Class inheritance

```
 99  class Vehicle():
100      def __init__(self, kind):
101          self.kind = kind
102
103      def what_am_i(self):
104          print(f'I am a {self.kind}.')
105
106      def fuel(self):
107          print('I use oil for fuel.')
```

inherited

inherited

overwritten

```
113  class Bicycle(Vehicle):
114      def fuel(self):
115          print('I am people-powered.')
116
117      def peddle_me(self):
118          print('You have to peddle fast.')
```

overwrites

adds

Class inheritance

```
113  class Bicycle(Vehicle):
114      def fuel(self):
115          print('I am people-powered.')
116
117      def peddle_me(self):
118          print('You have to peddle fast.')
```

```
In [29]: bike = Bicycle('bike')
    ...: bike.what_am_i()
    ...: bike.fuel()
    ...: bike.peddle_me()
I am a bike.
I am people-powered.
You have to peddle fast.
```

Arg comes from inherited __init__

Class inheritance

Python's base string class

```
126  class MyString(str):
127      def say_hello(self):
128          print('Hello world!')
129
130      def lower(self):
131          print("No, I don't want to be in lower case!")
```

```
In [39]: ms = MyString('Hello my name is Jeff.')

In [40]: ms.lower()
No, I don't want to be in lower case!

In [41]: ms.upper()
Out[41]: 'HELLO MY NAME IS JEFF.'

In [42]: ms.say_hello()
Hello world!
```