[301] Objects

Tyler Caraza-Harter

Learning Objectives Today

More data types

- tuple (immutable list)
- custom types: creating objects from namedtuple and recordclass

References

- Motivation
- “is” vs “==”
- Gotchas (interning and argument modification)

Read:

- **Downey Ch 10 ("Objects and Values" and “Aliasing")**
- **Downy Ch 12**

Today's Outline

New Types

- **tuple**
- namedtuple
- recordclass

References

- motivation
- unintentional argument modification
- “is” vs. “==”

Tuple Type

```
nums_list  = [200, 100, 300]
nums_tuple = (200, 100, 300)
```

if you use parentheses (round)
instead of brackets [square]
you get a tuple instead of a list

Tuple Type

```
nums_list  = [200, 100, 300]
nums_tuple = (200, 100, 300)
```

if you use parentheses (round)
instead of brackets [square]
you get a tuple instead of a list

What is a tuple?

Tuple Type

```
nums_list  = [200, 100, 300]
nums_tuple = (200, 100, 300)
```

Like a list

- for loop, indexing, slicing, other methods

Unlike a list:

- immutable (like a string)

Tuple Type

```
nums_list  = [200, 100, 300]
nums_tuple = (200, 100, 300)

print(nums_list[2])
print(nums_tuple[2])
```

Like a list

- for loop, **indexing**, slicing, other methods

Unlike a list:

- immutable (like a string)

Tuple Type

```
nums_list  = [200, 100, 300]
nums_tuple = (200, 100, 300)
```

```
print(nums_list[2])
print(nums_tuple[2])
```

both of these print 300

Like a list

- for loop, **indexing**, slicing, other methods

Unlike a list:

- immutable (like a string)

Tuple Type

```
nums_list  = [200, 100, 300]
nums_tuple = (200, 100, 300)

nums_list[0] = 22
nums_tuple[0] = 22
```

Like a list

- for loop, indexing, slicing, other methods

Unlike a list:

- **immutable** (like a string)

Tuple Type

```
nums_list  = [200, 100, 300]
nums_tuple = (200, 100, 300)

nums_list[0] = 22
nums_tuple[0] = 22
```

changes list to
[22, 100, 300]

Like a list

- for loop, indexing, slicing, other methods

Unlike a list:

- **immutable** (like a string)

Tuple Type

```
nums_list  = [200, 100, 300]
nums_tuple = (200, 100, 300)

nums_list[0] = 22
nums_tuple[0] = 22
```

changes list to
[22, 100, 300]

Crashes!

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Like a list

- for loop, indexing, slicing, other methods

Unlike a list:

- **immutable** (like a string)

Tuple Type

```
nums_list  = [200, 100, 300]
nums_tuple = (200, 100, 300)

nums_list[0] = 22
nums_tuple[0] = 22
```

changes list to
[22, 100, 300]

Crashes!

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Like a list

- for loop, indexing, slicing, other methods

Unlike a list:

- **immutable** (like a string)

Why would we ever want immutability?

1. avoid certain bugs
2. some use cases require it (e.g., dict keys)

Example: location -> building mapping

```
buildings = {
  [0,0]: “Comp Sci”,
  [0,2]: “Psychology”,
  [4,0]: “Noland”,
  [1,8]: “Van Vleck”
}
```

trying to use x,y coordinates as key

FAILS!

```
Traceback (most recent call last):
  File "test2.py", line 1, in <module>
    buildings = {[0,0]: "CS"}
TypeError: unhashable type: 'list'
```

Example: location -> building mapping

```
buildings = {
   (0,0): “Comp Sci”,
   (0,2): “Psychology”,
   (4,0): “Noland”,
   (1,8): “Van Vleck”
}
```

trying to use x,y coordinates as key

Succeeds!

(with tuples)

Today's Outline

New Types

- tuple
- **namedtuple**
- recordclass

References

- motivation
- unintentional argument modification
- “is” vs. “==”

Organizing Attributes

Often, we have **entities/objects** in programming with many **attributes**. E.g., a tornado. Or a **person**:

- first name, last name
- birth date
- SSN
- address
- phone number

Organizing Attributes

Often, we have **entities/objects** in programming with many **attributes**. E.g., a tornado. Or a **person**:

- first name, last name
- birth date
- SSN
- address
- phone number

One representation strategy: dictionaries

```
person = {
   “lname”: “Turing”, “fname”: Alan, …
}
```

Organizing Attributes

Often, we have **entities/objects** in programming with many **attributes**. E.g., a tornado. Or a **person**:

- first name, last name
- birth date
- SSN
- address
- phone number

One representation strategy: dictionaries

```
person = {
   “lname”: “Turing”, “fname”: Alan, …
}
print(person[“fname”] + “ “ + person[“lname”])
```

Organizing Attributes

Often, we have **entities/objects** in programming with many **attributes**. E.g., a tornado. Or a **person**:

- first name, last name
- birth date
- SSN
- address
- phone number

Problem with using dicts:

- it's verbose (always typing quotes)
- error prone (same attributes not enforced)

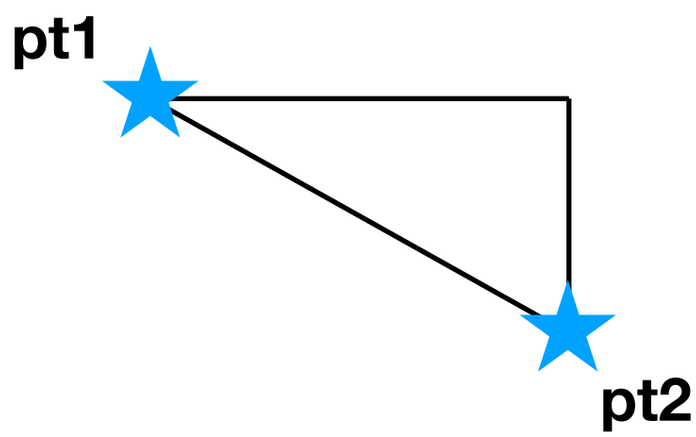
One representation strategy: dictionaries

```
person = {
   "lname": "Turing", "fname": Alan, …
}
print(person["fname"] + " " + person["lname"])
```

Tuples, with and without names

regular tuples (**remember** x then y)

```
pt1 = (50,60)
pt2 = (90,10)
distance = ((pt1[0]-pt2[0])**2 + (pt1[1]-pt2[1])**2) ** 0.5
```

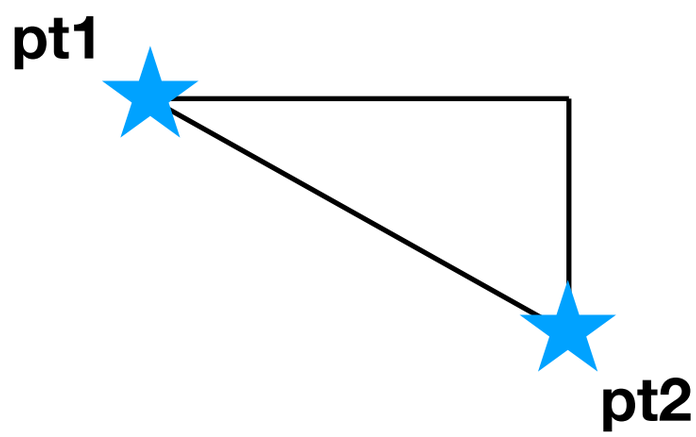


Tuples, with and without names

regular tuples (**remember** x then y)

pt1[0] is x

```
pt1 = (50,60)
pt2 = (90,10)
distance = ((pt1[0]-pt2[0])**2 + (pt1[1]-pt2[1])**2) ** 0.5
```



Tuples, with and without names

regular tuples (**remember** x then y)

pt1[0] is x

```
pt1 = (50,60)
pt2 = (90,10)
distance = ((pt1[0]-pt2[0])**2 + (pt1[1]-pt2[1])**2) ** 0.5
```

```
from collections import namedtuple
```

need to import namedtuple
(not there by default)

Tuples, with and without names

regular tuples (**remember** x then y)

pt1[0] is x

```
pt1 = (50,60)
pt2 = (90,10)
distance = ((pt1[0]-pt2[0])**2 + (pt1[1]-pt2[1])**2) ** 0.5
```

```
from collections import namedtuple

Point = namedtuple("Point", ["x", “y”])
```

Point is a new type

“Point” is the type’s name

A Point will have an x and y

Tuples, with and without names

regular tuples (**remember** x then y)

```
pt1 = (50,60)
pt2 = (90,10)
distance = ((pt1[0]-pt2[0])**2 + (pt1[1]-pt2[1])**2) ** 0.5
```

pt1[0] is x

```
from collections import namedtuple

Point = namedtuple("Point", ["x", “y”])
```

Point is a new type

“Point” is the type’s name

A Point will have an x and y

```
>>> L = list()
>>> type(L)
<class 'list'>

>>> type(list)
<class 'type'>

>>> type(Point)
<class 'type'>
```

Tuples, with and without names

regular tuples (**remember** x then y)

```
pt1 = (50,60)
pt2 = (90,10)
distance = ((pt1[0]-pt2[0])**2 + (pt1[1]-pt2[1])**2) ** 0.5
```

pt1[0] is x

```
from collections import namedtuple

Point = namedtuple("Point", ["x", “y”])
```

Point is a new type

“Point” is the type’s name

A Point will have an x and y

```
>>> L = list()
>>> type(L)
<class 'list'>

>>> type(list)
<class 'type'>

>>> type(Point)
<class 'type'>
```

Point is a now a datatype, like a list or dict.
Just like dict(…) and list(…) create new instances,
Point(…) will create new instances

Tuples, with and without names

regular tuples (**remember** x then y)

pt1[0] is x

```
pt1 = (50,60)
pt2 = (90,10)
distance = ((pt1[0]-pt2[0])**2 + (pt1[1]-pt2[1])**2) ** 0.5
```

```
from collections import namedtuple

Point = namedtuple("Point", ["x", "y"])

pt1 = Point(50,60)
```

x y

Tuples, with and without names

regular tuples (**remember** x then y)

pt1[0] is x

```
pt1 = (50,60)
pt2 = (90,10)
distance = ((pt1[0]-pt2[0])**2 + (pt1[1]-pt2[1])**2) ** 0.5
```

```
from collections import namedtuple

Point = namedtuple("Point", ["x", "y"])

pt1 = Point(50,60)
pt2 = Point(x=90, y=10)
```

x

y

Tuples, with and without names

regular tuples (**remember** x then y)

pt1[0] is x

```
pt1 = (50,60)
pt2 = (90,10)
distance = ((pt1[0]-pt2[0])**2 + (pt1[1]-pt2[1])**2) ** 0.5
```

```
from collections import namedtuple

Point = namedtuple("Point", ["x", "y"])

pt1 = Point(50,60)
pt2 = Point(x=90, y=10)

distance = ((pt1.x - pt2.x)**2 + (pt1.y - pt2.y) ** 2) ** 0.5
```

don’t need to remember anything (e.g., “x” is first)

Tuples, with and without names

regular tuples (**remember** x then y)

pt1[0] is x

```
pt1 = (50,60)
pt2 = (90,10)
distance = ((pt1[0]-pt2[0])**2 + (pt1[1]-pt2[1])**2) ** 0.5
```

```
from collections import namedtuple

Point = namedtuple("Point", ["x", "y"])

pt1 = Point(50,60)
pt2 = Point(x=90, y=10)

distance = ((pt1.x - pt2.x)**2 + (pt1.y - pt2.y) ** 2) ** 0.5
```

```
>>> pt1.x = 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: can't set attribute
```

note that nametuples are also immutable

Today's Outline

New Types

- tuple
- namedtuple
- **recordclass**

References

- motivation
- unintentional argument modification
- “is” vs. “==”

recordclass example

```
>>> from recordclass import recordclass
```

module is recordclass

so is function

recordclass example

```
>>> from recordclass import recordclass
>>> Point = recordclass("Point", ["x", "y"])
```

```
Point = namedtuple(“Point", ["x", "y"])
```

recordclass example

```
>>> from recordclass import recordclass
>>> Point = recordclass("Point", ["x", "y"])
>>> pt1 = Point(0,0)
>>> pt1
Point(x=0, y=0)
```

recordclass example

```
>>> from recordclass import recordclass
>>> Point = recordclass("Point", ["x", "y"])
>>> pt1 = Point(0,0)
>>> pt1
Point(x=0, y=0)
>>> pt1.x = 5
>>> pt1.y = 6
```

mutations

recordclass example

```
>>> from recordclass import recordclass
>>> Point = recordclass("Point", ["x", "y"])
>>> pt1 = Point(0,0)
>>> pt1
Point(x=0, y=0)
>>> pt1.x = 5
>>> pt1.y = 6
>>> pt1
Point(x=5, y=6)
```

recordclass example

```
>>> from recordclass import recordclass
>>> Point = recordclass("Point", ["x", "y"])
>>> pt1 = Point(0,0)
>>> pt1
Point(x=0, y=0)
>>> pt1.x = 5
>>> pt1.y = 6
>>> pt1
Point(x=5, y=6)
```

Note: recordclass does not come with Python.
You must install it yourself.

Aside: installing packages

There are many Python packages available on PyPI

- https://pypi.org/
- short for Python Package Index

Installation example (from terminal):

```
pip install recordclass
```

Aside: installing packages

There are many Python packages available on PyPI

- https://pypi.org/
- short for Python Package Index

Installation example (from terminal):

```
pip install recordclass
```

Anaconda is just Python with a bunch of packages related to data science and quantitative work pre-installed.

Today's Outline

New Types

- tuple
- namedtuple
- **recordclass** ← **mutable equivalent of a namedtuple**

References

- motivation
- unintentional argument modification
- “is” vs. “==”

Today's Outline

New Types

- tuple
- namedtuple
- recordclass

References

- motivation
- unintentional argument modification
- “is” vs. “==”

Mental Model for State (v1)

Code:

```
x = “ha” * 3
y = x
…
```

State:

x

y

Mental Model for State (v1)

Code:

```
x = “ha” * 3
y = x
…
```

State:

x	hahaha
y	

Mental Model for State (v1)

Code:

```
x = “ha” * 3
y = x
…
```

State:

x	hahaha
y	hahaha

Today's Outline

New Types

- tuple
- namedtuple
- recordclass

References

- **motivation**
- unintentional argument modification
- “is” vs. “==”

Today's Outline

New Types

- tuple
- namedtuple
- recordclass

References

- motivation
- **unintentional argument modification**
- “is” vs. “==”

Today's Outline

New Types

- tuple
- namedtuple
- recordclass

References

- motivation
- unintentional argument modification
- **“is” vs. “==”**