Data Types

Intermediate Application Development

Otago Polytechnic
Dunedin, New Zealand
Kaiako: Tom Clark

Types in Python

Python has most of the types with which you're already familiar. Its built in collection types are a bit more powerful and flexible than those seen in other languages, so we'll mostly talk about them. First, though, we need to talk about three core ideas:

- variables
- mutabilty vs. immutability
- equality vs. identity

Variables

A Python variable can be regarded as a reference to a value.

```
a = 4
```

`a` is a variable that now refers to the integer value `4`.

```
a = 'cat'
```

Now `a` refers to the string value `'cat'`.

```
b = a
```

`b` is a variable that refers to the same value that `a` does (for now).

```
a = 4
```

Now `a` refers to `4` again. To what does `b` refer?

Immutable values

A lot of the primitive values in Python are *immutable*, i.e., they can't be changed. Integers are an obvious example.

```
a = 4
a = 5
```

We did't change the value of `4` here. We just made `a` point to a different value.

Numeric types, strings, and some other types are immutable, but others are not.

Mutable values

One example of a mutable type is a *list*. (We'll talk more about these in a few minutes.)

```
a = [1, 2, 3]
```

`a` is a list that contains the integers 1, 2, and 3.

```
a.append(4)
```

Now the list contains 1, 2, 3, and 4 - but it's still the **same** list.

```
b = a
```

`b` refers to the same list that `a` does.

```
a.append(5)
```

Now what is the value of `b`?

Equality vs. Identity

There are two ways to compare objects in Python. `a == b` means that

`a` and `b` have the same *value*.

`a is b` means that `a` and `b` refer to the same *object*.

```
> a = 4
> b = a
> a == b
True   # Good!
> a is b
True   # Also good
```

Equality vs. Identity

Remember:

`a == b` means that `a` and `b` have the same *value*.

`a is b` means that `a` and `b` refer to the same *object*.

```
> a = 5
> b = 5
> a == b
True   # No suprise
> a is b
True   # Hmmm...
> a = 55555
> b = 55555
> a == b
True   # Whew
> a is b
False  # Wat?
```

Programming Activity

1. Pull the course materials repo.
2. Create a new branch, `02-practical` in your practicals repo.
3. Add a subdirectory, `02-practical` and copy `02-practical.ipynb` from the class materials into it.
4. Open a shell, cd to this directory, and run `jupyter notebook` to open the notebook. Complete the first questions.
5. We will discuss results in 20ish minutes.

Lists

- Ordered collection of values
- Can be indexed by integer position
- Mutable

```
nums = [1, 2, 3, 4, 5] # Homogeneous
hetero = [1, 'C#', True, 2, 'Java'] # Heterogeneous
print(type(nums)) # <class ?list?>
```

Tuples

- Ordered collection of values
- Can be indexed by integer position
- Immutable, BUT can contain mutable values, e.g., lists

```
nums = (1, 2, 3, 4, 5) # Homogeneous
hetero = (1, 'C#', True, 2, 'Java') # Heterogeneous
print(type(nums)) # <class 'tuple'>
```

Sets

- Unordered collection of values
- Doesn't contain duplicate values
- Mutable

```
nums = {1, 2, 3, 4, 4} # Homogeneous
hetero = {1, 'C#', True, 2, 2} # Heterogeneous
print(type(nums)) # <class 'set'>
print(nums) # {1, 2, 3, 4}
print(hetero) # {'C#', 1, 2}
```

Why doesn't `True` appear in the values when we print `hetero`?

Dictionaries

- Unordered collection of key/value pairs
- Mutable
- Keys can be any immutable object

```
ig_user_one = {'username': 'john_doe', 'active': False, 'f
ig_user_two = {'username': 'jane_doe', 'active': True, 'fo
print(type(ig_user_one)) # <class 'dict'>
print(ig_user_one['username']) # john_doe
print(ig_user_two['followers']) # 500
```

Strings

- Ordered collection of unicode character values
- Can be indexed by integer position
- Immutable
- They print nicely

```
st = 'I have a cat named Lola.'
print(type(st)) # <class 'str'>
```

A key point

Python's collection types are very flexible and powerful. They come with many built-in methods. You can solve many common problems using just these types.