Working with basic types and collections

- Objectives
  - Convert between types
  - Work with collections
  - Use slice operations

Fundamental types

- There are a few types in Python that you must be fluent in
- These include:
  - numbers
  - strings
  - dates and times
  - collections

Numerical types [details]

- Defining numerical types

```
n = 1                          # int
a = 1000000000000000000        # int
x = 0.0                        # float
y = 5 * x                      # float
```

http://docs.python.org/2.4/lib/typesnumeric.html
https://docs.python.org/3/library/stdtypes.html#numeric-types-int-float-complex

Conversions [between numerical types]

- You can explicitly convert between numerical types

```
n = 14
x = float(n)
m = int(x)

n => 14
x => 14.0
m => 14
```

Conversions [from strings]

- More common is to convert from strings

```
n = int("14")
x = float("14.7")

n => 14
x => 14.7
```

Strings [defining]

- All strings in Python 3 are Unicode and immutable
- Can be defined with
  - Double quotes
  - Single quotes
- Have escape characters similar to C++ / C#
  - However, you can treat them as 'raw' strings with **r** prefix

```
s = "is a string"
t = 'is also a string'

para = "strings can\t\thave escape\nchars including\nnewlines"
print(para)
# strings can         have escape
# chars including
# newlines

raw = r"I wouldn\t escape this\n."
print(raw) # I wouldn\t escape this\n.
```

Strings [multiple lines]

- There are several techniques for spanning lines
  - Any continuation char: \
  - """ (three quotes) or ‘‘‘ (three single quotes)

```
crossLine1 = "This spans" + \
             "a code line"



crossLine2 = \
"""
This string is designed to span
lines and be exactly as you see
it (including line breaks)
"""
# this one includes line breaks.
```

Strings [methods]

- Strings have many utility methods including:
  - upper()
  - lower()
  - find() / index()

```
"Some string".
```

capitalize(self)	str
casefold(self)	str
center(self, width, fillchar)	str
count(self, sub, start, end)	str
encode(self, encoding, errors)	str
endswith(self, suffix, start, end)	str
expandtabs(self, tabsize)	str
find(self, sub, start, end)	str
format(args, kwargs)	str
format_map(self, mapping)	str

Press Ctrl+Period to choose the selected (or first) suggestion and insert a dot afterwards >>

Strings [formatting]

- **string.format()** is a powerful string construction method.

```
"Hello {0}, today is {1}. Right {0}?“
    .format("Michael", "Monday"))
# Hello Michael, today is Monday. Right Michael?

"{0:,} is pretty big!".format(1234567890)
# 1,234,567,890 is pretty big!

"You can name your args {jeff} and {tony}!"
    .format(jeff="bigj", tony="t-boy")
# You can name your args bigj and t-boy!

"v3.1 added empty {} and {}!".format("placeholders", "such")
# v3.1 added empty placeholders and such!
```

Strings [miscellanea]

- String length is computed via **`len(txt)`** method.
- Strings can be indexed
  - **`txt[2]`** => 3rd character (zero based)
  - **txt[-3]** => 3rd from last character (-1 based)
- They are 'mathy'
  - **`"hi" * 2 + "bye"`** => "hihibye"
- They can be combined via + or just adjacency
  - `"Combine " + "this" => "Combine this"`
  - `"Combine " "this" => "Combine this"`

Dates and times

- Python has support for dates, times and timespans
- Defined within the **`datetime`** module
  - `from datetime import date`
  - `from datetime import time`
  - `from datetime import datetime`

```
today = date.today()
print("Today is {month}/{day}/{year}"
    .format(month=today.month, day=today.day, year=today.year))

# Prints: Today is 11/25/2013

now = datetime.now()
print("Right now it's {0}h:{1}m:{2}.{3}sec"
        .format(now.hour, now.minute, now.second,
        now.microsecond//1000))

# Prints: Right now it's 16h:20m:5.867sec
```

Dates and times [timespans]

- **timedelta** class manages time spans.
- Defined within the **datetime** module
  - `from datetime import timedelta`
- Result of subtraction between two datetimes
  - `dt = t1 - t0 # dt is a timedelta`

```
dt = timedelta(hours=1, minutes=5)

now = datetime.now()
later = now + dt

print("Now it's {0} but will be {1}.".format( now, later))

# Prints:
# Now it's 2013-11-25 16:27:22 but will be 2013-11-25 17:32:22.
```

Dates and times [parsing]

- Parse text with **`datetime.strptime`**

```
txt = "Monday, November 21, 2013"
day = datetime.strptime(txt, "%A, %B %d, %Y")

print(day)
# 2013-11-21 00:00:00
```

There are many options for the format string:
http://docs.python.org/3.4/library/datetime.html#strftime-strptime-behavior

Collections

- Python has a rich set of collection classes
  - Lists []
  - Sets ()
  - Dictionaries {}
  - Tuples ()
- The interfaces of each is generally consistent
- Python idioms rely heavily on collections

Lists

- Lists are the most fundamental collection type in Python
- Lists are essentially Python's array type
- Lists are defined using the **list** class or **[ ]** (square brackets)

```
numbers = []        # an empty list
numbers = list()    # another empty list
numbers = [1,2,3]   # a list with items

# lists can be heterogeneous
numbers = [1,2,3,"not a number"]
```

Lists [accessing values]

- Lists are iterable and indexable
  - Forward **Indexes** are zero-based
  - **Backwards Indexes** are negative-one-based
  - **for** loops pull out the values one at a time

```
num = [1,2,3,4,5]

first = num[0]              # value = 1
last  = num[len(num) - 1] # value = 5
Last  = num[-1]             # value = 5

for n in num:
    even_text = "even" if n % 2 == 0 else "odd"
    print("{0} is {1}.".format(n, even_text))
```

Lists [building lists]

- Lists can be built-up dynamically
  - one item at a time via **`list.append()`**
  - via unions (**`+`**)
  - via **`list.extend()`**

```
num = []
num.append(7)
num.append(11) # num = [7, 11]

num = [7, 11] + [13, 17, 19]
num.extend([23,97])

# num = [7, 11, 13, 17, 19, 23, 97]
```

Lists [removing items]

- List items can be changed and removed

Indexer is read/write

Removes first occurrence by value

Removes last

```
num = [7, 11, 14, 17, 19, 17, 23, 97]

num[2] = 4      # [7, 11, 4, 17, 19, 17, 23, 97]
num.remove(17) # [7, 11, 4, 19, 17, 23, 97]

v = num.pop()
# v   => 97
# num => [7, 11, 4, 19, 17, 23]

del num[2] # [7, 11, 19, 17, 23]
```

removed by index via **del** keyword

Slicing

- Python has a technique for dissecting **strings** and **lists**
- Takes the form of
  - **`item[ startIndex : endIndex : step ]`**

```
num = [7, 11, 13, 17, 19]

num[2:4]  # [13, 17]
num[2:]   # [13, 17, 19] omit end index = len(num)-1
num[:3]   # [ 7, 11, 13] omit start index = 0

num[-2:]  # [17, 19] reverse

s = "This also works on strings"
s[-10:]    # on strings
```

Collections [sets]

- Sets are an unordered collection of distinct objects
  - Supports set theoretic operations

```
# defining sets
s = set() # not { }, {} is a dictionary.
s = {1,2,2,2,5}

# modifying sets
s.add(3)
s.add(3)

print(s) # prints {1, 2, 3, 5}
```

Collections [dictionaries]

- Dictionaries map hashable values (keys) to arbitrary objects (values).

```
# defining dictionaries
d = dict()
d = {}
d = {"one": "monday",
     "two": "tuesday",
     "three": "wednesday"}

# adding items
d["four"] = "thursday"

# checking for items
"three" in d # True
"seven" in d # False

# accessing items
d["three"] # "thursday"
d["seven"] # KeyError exception
```

Collections [tuples]

- Typically used to store collections of heterogeneous data

```
# create a tuple
t = (1, 2, "orange")
t[1] # => 2

# assignment to multiple variables
x, y, color = t # x=1, y=2, color=orange
```

Tuples are often return values from methods

```
for (index, item) in enumerate(["first", "middle", "last"]):
    print("{}: {}".format(index, item))

# prints
0: first
1: middle
2: last
```

Summary

- Strong support for scientific / numerical operations
- Variety of numerical types: integers, floats
- Convert between types using integer(), str(), etc.
- All strings are Unicode in Python 3
- Strings support a clean format style
- There are 4 fundamental collection types: lists, sets, dictionaries, and tuples
- Slicing allows us to work with subsets of collections