Intro to Python

March 14, 2017

1 Intro to Python

1.1 System Administrators Welcome

1.1.1 Day 1, morning session

Naomi Ceder, @naomiceder

```
import antigravity
```

1.2 Introductions

I'm really old

1.2.1 Me

- Naomi Ceder
- Python since 2001 (v2.2)
- Quick Python Book, 2nd & 3rd eds.
- Python Software Foundation
- Lead Dev, System Architect, etc.

1.3 You

- What do you do?
- What coding experience do you have?
- What are your repetitive hassles and time sinks?
- What problems do you want/hope to solve with code?

1.4 What we'll do, Day 1

Morning - baby steps

- Introduction
- Data types
- Basic syntax

Afternoon

- Basic programs and style
- Functions & modules

1.5 What we'll do, Day 2

Morning

- Basic file handling
- Advanced features

Afternoon

- Packages
- Objects
- Standard library

1.6 How this works

- I talk
- You question & comment
- I code
- You code
- I coach
- Post-Its?

1.7 What Python Does Well

- Readable
- Powerful
- High level
- Batteries included
- "Low floor, high ceiling"
- Prototyping, 'glue', sysadmin, web apps, data handling & exploration, scientific,

1.8 What Python doesn't do so well

- Mobile
- Concurrency (the GIL)
- Raw speed

1.9 2 vs 3

- What's the fuss?
- For 15 years (1995 - 2009), Python versions were backward compatible
- Python 3, after years of planning, broke backward compatibility
- Prophecies of doom were in the air...

1.10 2 vs 3

- What changed?
- strings/unicode -> bytes/strings
- Libraries reorganized, cleaned up

- print -> print()
- Division
- Only one int
- Comparisons
- Views and iterators

1.11 2 vs 3

- What changed?
- strings/unicode -> bytes/strings
- Libraries reorganized, cleaned up
- print -> print()
- Division
- Only one int
- Comparisons
- Views and iterators

1.11.1 Is 3 better?

- Yes, it actually is

1.11.2 Why is 2 still so common?

- Installed codebase
- Lack of tests
- inertia

1.12 Python in the shell

You'll be using the standard Python shell. To help present, I'm using a Jupyter notebook. Here's what a Python shell typically looks like.

```
In [ ]: Python 2.7.12 (default, Nov 19 2016, 06:48:10)
        [GCC 5.4.0 20160609] on linux2
        Type "help", "copyright", "credits" or "license" for more information.
        >>> 4
        4
        >>> 5 + 6
        11
        asdf
        Traceback (most recent call last):
          File "<stdin>", line 1, in <module>
        NameError: name 'asdf' is not defined
```

1.13 Exercise: Exploratory messing around

Test out the shell - enter words, numbers, and math operations and see what happens. Can you enter something without getting an error message?

1.14 Data types

- Python and Types
- Everything is an object
- Objects are strongly typed
- “Variables” are not typed (more late

1.15 “Simple” Data Types

Name
Explanation
Example
int
integer, at least 32 bit
1, 1234567890
long
unlimited
1L, 12345678901234567890L
float
floating point, “double”
0.5, 1234.567
complex
complex number
2+1J
bool
true or false boolean value
True, False
None
Not set
None

Some “built-in” functions: * int() - converts floats (and strings) to integerts * bin() - converts integer to binary representation * oct() - converts integer to octal representation * hex() - converts integer to hex representation * long() - works the same as int() but yields a long * float() - converts ints, longs, and correctly formatted strings to floats * complex() - converts a real and imaginary part (and correctly formatted strings) to complex * bool() - converts most values to either True or False * abs() - absolute value of int, long, float

```
In [1]: >>> 1234567890
        1234567890
        >>> 12345678901234567890
        12345678901234567890L
        >>> 1l
        1L
        >>> 4/2L
        2L
        >>> 3.3
        3.3
        >>> 1+3J
        (1+3j)
        >>> True
        True
        >>> False
        False
        True

Out[1]: True
```

```
In [15]: >>> hex(10)
         '0xa'
         >>> bin(0xa)
         '0b1010'
         >>> oct(10)
         '012'
         >>> long(10)
         10L
         >>> bool(0)
         False
         >>> bool(-1)
         True
         >>> float(10)
         10.0
         >>> abs(-10)
         10

Out[15]: 10
```

1.15.1 A caveat on floats

Floats are not exactly decimal numbers, since they're based on binary values. So comparisons of floats are inherently risky and should be avoided.

More explanation here

```
In [93]: print 0.1 + 0.2 == 0.3

         from decimal import Decimal
         print Decimal(0.1) + Decimal(.2)
         print Decimal(0.3)
```

```
False
0.3000000000000000166533453694
0.299999999999999988897769753748434595763683319091796875
```

1.16 Common Python operators

- + - addition
- - - subtraction

- * - multiplication
- / - division, integer divsion - **if both dividend and divisor are integers, integer division will be performed and any fraction will be discarded**

Multiplication and division take precedence, but operations can be grouped with parentheses

```
In [16]: >>> 3 + 2
         5
         >>> 3 - 2
         1
         >>> 3 * 2
         6
         >>> 3 / 2
         1
         >>> 3.0 / 2
         1.5
         >>> 3 + 2 * 4
         11
         >>> (3 + 2) * 4
         20

Out[16]: 20
```

1.17 “Variables”

- Variables are labels
- They are not buckets
- They refer to objects,
- which are created when first set
- Variables can refer to different types/objects
- Objects can’t change type

```
In [23]: >>> a = 1
         >>> b = 2
         >>> c = b
         >>> c
         2
         >>> a + b
         3
         >>> b + c
         4
         >>> b = None
         >>> b + c #----> error


        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)
```

```
    <ipython-input-23-7ca45c674634> in <module>()
      9 4
     10 b = None
---> 11 b + c #----> error


    TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

1.18 print

- not a function in Python 2 (but it is in Python 3)
- elements separated by commas - no line feed, just space
- goes to stdout

```
In [12]: print 1, 2

         print 1 + 2
         print 3 + 4

         print 1 + 2,
         print 3 + 4

         a = 1 + 2
         print a
         print 'a'
```

```
1 2
3
7
3 7
3
a
```

1.19 Exercise: Python as a calculator

The formula for converting Farhenheit to Celsius is Celsius equals Fahrenheit minus 32 times 5/9; or going the other way, Fahrenheit = Celsius times 9/5 plus 32. Using variables for F and C create and test formulas to convert both ways that end up looking something like this:

```
>>> F = <some integer Fahrenheit temperature>
>>> C = <your formula>
>>> print C
<the correct temperature in Celsius>

>>> C = <some integer Celsius temperature>
>>> F = <your formula>
>>> print F
```

```
<the correct temperature in Fahrenheit>
```

Test with the same temperatures in both directions.

```
In [20]: F = 212
         C = F-32 * 5.0/9
         print C

         F = C * 9/5.0 + 32
         print F
```

```
194.222222222
381.6
```

1.20 Less common operators

- % - modulus, remainder
- divmod() - quotient, remainder
- **, pow() - power
- += adds the second operand to the first

```
In [ ]:
```

```
In [8]: >>> 5 / 3
        1
        >>> 5 % 3
        2
        >>> divmod(5, 3)
        (1, 2)
        >>> 3 ** 2
        9
        >>> pow(3, 2)
        9
        a = 1
        print a
        a += 2
        print a
        a = a + 2
        print a
```

```
1
3
5
```

1.20.1 Strings, part 1

- Sequence of characters

- strings are immutable
- marked by single quotes, double quotes, triple single quotes, triple double quotes
- triple quoted version can span lines
- strings by themselves are not executed -> docstrings

```
In [1]: print "hello", "world"

        print "hello"
        print "world"

        print "hello",
        print "world"
```

```
hello world
hello
world
hello world
```

```
In [ ]: "hello"
        'hello'
        """hello
        world"""
        '''hello
        world'''
```

1.21 Special characters and escapes

- triple quoted strings can contain single and double quotes
- double quoted strings can contain single quotes and vice versa
- quotes and other special characters can be escaped with a ″

```
In [2]: "I'm here"
        'She said "what?"'
        """I'm saying "stop" """
        "\t is a tab"
        "\n is a newline"
        "\" is a double quote"
        "\\ is a \\"
```

```
Out[2]: '\\ is a \\'
```

1.21.1 raw_input()

- raw_input() gets a value **as a string** from the user
- parameter can be a prompt
- you must transform the string into other types - int(), float(), etc

```
In [15]: c = (int(raw_input("F?")) - 32) * 5.0/9
         print "centigrade =", c
```

```
F?32
centigrade = 0.0
```

1.22 Getting help

- dir() - listing of what is in the namespace, good for quick reference
- help() - fuller help on any object (q to exit in shell)

```
In [18]: print dir(int)
```

```
['__abs__', '__add__', '__and__', '__class__', '__cmp__', '__coerce__', '__delattr__', '__div__'
```

```
In [27]: bool(int("0"))
```

```
Out[27]: False
```

1.23 Boolean Comparison operators

`<, >, <=, >=, ==, !=, in`

1.24 Truthiness in Python

False
`False, 0, [], (), None`, empty (in general)
True
Everything else

1.25 Blocks in Python

- indentation groups code blocks
- use spaces, not tabs
- NEVER mix spaces and tabs

1.26 `if` statements

- `if` followed by boolean expression
- `elif` followed by boolean expression, checked if previous condition not true
- `else` if no previous condition is true

```
In [31]: x = int(raw_input("Enter a number: "))
         if x > 8:
             print "High"
             print "Actually, it's really high"
         elif x >=6:
             print "A little high"
         elif x > 8:
             print "Low"
```

```
        elif x <=4:
            print "A little low"
        else:
            print "Correct"

        print "Game over"
```

```
Enter a number: 9
High
Actually, it's really high
Game over
```

1.27 Exercise: Guess the number

Use raw_input and if statements to make a number guessing game where the computer chooses a number from 1 to 10 and the you try to guess it. The game should tell you if you are right, high, or low.

NOTE: to make the program select a random number you will need to do 2 things:

1. **load the random module** - `import random`
2. **for every run** you will need to select a number - `number = random.randint(1, 11)`

```
In [68]: import random
         number = random.randint(1,10)
         print number

         print __name__
```

```
6
__main__
```

```
In [2]: import random

        number = random.randint(1, 11)
        guess = int(raw_input("Guess? "))
        if guess == number:
            print "Correct!"

        elif guess < number:
            print "Low"

        else:
            print "High"
```

```
Guess? 3
High
```

1.28 Programs and basic code structure

- Comments
- Docstrings
- "Dunder" main
- "sh-bang" line (optional)

```
In [69]: #!/bin/env python
         """
             This is a docstring for the entire program

             You can put details about the purpose, usage, parameters,
             date, author, license, etc here.

             In general, you should use docstrings as documenation,
             and comments should be used to explain tricky bits of code

         """

         # this is the "dunder" main part
         if __name__ == "__main__":
             """ This is a docstring for the main part.
                 Not much need in a simple program, but very handy with more
                 complex scripts"""

             print "STOP! Who would cross the Bridge of Death must answer" + \
                   " me these questions three, ere the other side he see."

             name = raw_input("What is your name? ")
             quest = raw_input("What is your quest? ")
             color = raw_input("What is your favorite color? ")
             if color == "blue":
                 print "You may pass!"
             else:
                 print "Arrrghhhh!!!"
```

```
STOP! Who would cross the Bridge of Death must answer me these questions three, ere the other si
What is your name? Gawain
What is your quest? The Holy Grail
What is your favorite color? blue
You may pass!
```

1.29 Exercise: Start using programs

1. Open a file in your text editor of choise
2. Save the file with a `.py` extension

3. Enter your guess program in a style similar to the example above
4. Run the program from the command line with `python my_program.py`

2 Intro to Python

2.1 System Administrators Welcome

2.1.1 Day 1, afternoon session

Naomi Ceder, @naomiceder

2.2 "Pythonic" code

- Zen of Python (PEP-20) - `import this`
- PEP-8 - https://www.python.org/dev/peps/pep-0008/
- Elements of Python Style - https://github.com/amontalenti/elements-of-python-style/blob/master/README.md#the-elements-of-python-style
- Linters - pep8, flake-8, etc

```
In [1]: import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

2.3 lists

- sequence of elements (can be of any types, mix and match)
- accessed by index, starting with 0
- negative indexes count down from the end
- square brackets []

2.4 tuples

- immutable sequence, hashable
- fewer methods
- parentheses ()
- for single item tuple use trailing comman - (1,)

```
In [71]: a_list = [1, 2, 3]
         print a_list

         print a_list[-1]

         a_tuple = (1, 2, 3)
         print a_tuple

         single_item_tuple = (1,)
         print single_item_tuple
         print dir(tuple)
```

```
[1, 2, 3]
3
(1, 2, 3)
(1,)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__', '__format__', '__ge
```

2.5 for loops

- loop over a sequence (e.g., a list or tuple or string, or anything that is a sequence)
- for counting use range() and xrange()

```
for variable in sequence:
    do something
```

```
In [79]: for item in xrange(1,6):
             print item

         print xrange(1,6)
```

```
1
2
3
4
5
xrange(1, 6)
```

2.6 "Counting" loops

In many languages a `for` loop is used to do something for a specific number of times. In other words, in those languages a `for` loop counts, while in Python it moves over a sequence.

The example above could also be written as a counting loop by using the `range` function, as in the example below. Note however, that by default `range` starts counting from 0.

```
In [83]: for item in range(5):
             print item

         # make range start counting from 1
         #for item in range(1, 6):
         #    print item

         number = int(raw_input("enter a number"))
         for x in range(number+1):
             print x, "squared is", x * x
```

```
0
1
2
3
4
enter a number5
0 squared is 0
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

2.7 `while` loops in Python

- loop while a condition is true
- wider range of situations, less predictable, e.g. user input

Steps for loop survival 1. set loop variable 2. check loop variable 3. change loop variable so that it eventually makes the condition false

```
In [ ]: is_quit = 'n'     # setting loop variable
        while is_quit != "y":      # checking loop variable
            print "not quitting"
            is_quit = raw_input("Enter y to quit, any other key to continue: ")    #  changing lo
```

```
not quitting
```

2.8 More loop keywords

- continue
- break
- else

```
In [49]: x = 0
         while True:
             if x > 10:
                 break
             x += 1
             print x
```

```
1
2
3
4
5
6
7
8
9
10
11
```

```
In [87]: for x in 1, 2, 3, 4, 5:
             if not x % 2:
                 continue
             print x
```

```
1
3
5
```

```
In [5]: for x in  1, 2, 3, 4, 5:
            print x
        else:
            print "in else block, loop ended normally"

        for x in  1, 2, 3, 4, 5:
            if x > 3:
                break
            print x
        else:
            print "in else block, loop ended normally"
```

```
1
2
```

```
3
4
5
in else block, loop ended normally
1
2
3
```

2.9 Exercise : Make the guessing game a real game

Using a while loop, make the guessing game we created above into a proper game that keeps asking for guesses until the right answer is given or the user enters "quit".

```
In [ ]: """ guessing game
            computer picks a random number from 1-10
            user tries to guess
            if wrong, tell user 'high' or 'low'
            if user enters quit, quit
        """
        import random
        number = random.randint(1,11)

        # What follows is pseudocode...

        # get a guess
        # repeat as long as guess != number
        while True:
            guess = raw_input("enter guess?")
            if guess == "quit":
                print "Quit"
                break
            guess = int(guess)
            # if guess is low, say 'low'
            if guess < number:
                print 'low'
            # if guess is high, say 'high'
            elif guess > number:
                print "high"
            # if guess is right, say 'correct'
            else:
                print "equal"
                break
```

```
In [ ]: """ guessing game full final version
            computer picks a random number from 1-10
            user tries to guess
```

```
    if wrong, tell user 'high' or 'low'
    if user enters quit, quit
"""
import random
number = random.randint(1,11)


if __name__ == '__main__':

    # get a guess
    # repeat as long as guess != number
    while True:
        guess = raw_input("enter guess?")
        if guess == "quit":
            print "Quit"
            break
        guess = int(guess)
        # if guess is low, say 'low'
        if guess < number:
            print 'low'
        # if guess is high, say 'high'
        elif guess > number:
            print "high"
        # if guess is right, say 'correct'
        else:
            print "equal"
            break
```

2.10 List methods

- pop(slot) - remove and return the value
- remove(slot) - remove
- del list[slot] - also removes
- insert(slot, value) - inserts value into slot, makes room
- append(value) - adds to list
- extend(list2), + list2 - adds list2 to list
- index(value) - returns position of callue in list, ValueError if not found
- count(value) - returns number of times value occurs in list
- sort - sorts list in place, no return
- reverse - reverses list in place, no return
- * number - repeats list number times

```
In [28]: a_list = [3, 5, 1, 2, 6, 9, 8, 7, 4, 0]
         print a_list
         print "popping slot 1"
         a_list.pop(1)
         print a_list
         print "deleting slot 1"
         del a_list[1]
         print a_list

         print "inserting 1 into slot 1"
         a_list.insert(1, 1,)
         print a_list
         print "inserting 5 into slot 1"
         a_list.insert(1, 5)
         print a_list
         print a_list.index(6)
         print a_list.index(10)
```

```
[3, 5, 1, 2, 6, 9, 8, 7, 4, 0]
popping slot 1
[3, 1, 2, 6, 9, 8, 7, 4, 0]
deleting slot 1
[3, 2, 6, 9, 8, 7, 4, 0]
inserting 1 into slot 1
[3, 1, 2, 6, 9, 8, 7, 4, 0]
inserting 5 into slot 1
[3, 5, 1, 2, 6, 9, 8, 7, 4, 0]
4
```

```
        ValueErrorTraceback (most recent call last)

        <ipython-input-28-2ba3c3fadcf7> in <module>()
         15 print a_list
         16 print a_list.index(6)
    ---> 17 print a_list.index(10)


        ValueError: 10 is not in list
```

```
In [30]: a_list = [3, 5, 1, 2, 6]
         b_list = [9, 8, 7, 4]
         a_list.extend(b_list)
         print a_list
         a_list.append(0)
         print a_list

         print a_list.count(6)
         a_list.sort()
         print a_list
         a_list.reverse()
         print a_list

         print a_list * 2

         a_list = [3, 5, 1, 2, 6, 9, 8, 7, 4, 0]
         print a_list
         print sorted(a_list)
         print reversed(sorted(a_list))
         print list(reversed(sorted(a_list)))
```

```
[3, 5, 1, 2, 6, 9, 8, 7, 4]
[3, 5, 1, 2, 6, 9, 8, 7, 4, 0]
1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
[3, 5, 1, 2, 6, 9, 8, 7, 4, 0]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
<listreverseiterator object at 0x7f8508af2a10>
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

2.11 List slicing

- a “slice” is a starting index, a colon, an end limit
- steps - (optional) another colon and a step value
- negative numbers count from the end
- negative steps count down
- the end limit is a line that won’t be crossed
- empty indexes mean "all the way to the end"
- slices can be changed - e.g. a_list[2:4] = []
- a “full” slice will make a shallow copy [:]

```
In [36]: a_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
         print a_list[:5]
         print a_list[5:]
         print a_list[-1::-1]

         a_list[3:4] = [14, 15, 16, 17]
         print a_list

         a_list[3:7] = [12]
         print a_list

         a_list[3:4] = []
         print a_list
```

```
[0, 1, 2, 3, 4]
[5, 6, 7, 8, 9]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
[0, 1, 2, 14, 15, 16, 17, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 12, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 4, 5, 6, 7, 8, 9]
```

2.12 Reminder about variables

Variables are references not containers... copying just adds another reference to the same object, it does **not** create a new object.

```
In [37]: a_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
         b_list = a_list
         print b_list
         a_list[0] = -1
         print a_list

         print "a_list", id(a_list)
         print "b_list", id(b_list)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[-1, 1, 2, 3, 4, 5, 6, 7, 8, 9]
a_list 140209493608928
b_list 140209493608928
```

2.13 More built-in functions

- len(sequence) - returns length
- max(sequence), min(sequence) - returns max, min values
- sorted(sequence) - returns sorted list of sequence
- reversed(sequence) - returns iterator of reversed sequence

See https://docs.python.org/2.7/library/functions.html

```
In [42]: x = [5, 2, 3, 1, 4]
         print len(x)
         print max(x), min(x)
         print sorted(x)
         print reversed(x)
```

```
5
5 1
[1, 2, 3, 4, 5]
<listreverseiterator object at 0x7f85093f5d50>
```

2.14 Dictionaries

- key, value storage
- key is hashed
- keys must be immutable, hashable (no lists, dictionaries, sets)
- HashMap (Java), Hash (Ruby), Object (Javascript)
- Marked by curly braces - {}, colons separate key and value, commas separate elements
- **Accessed using square brackets**
- Are **NOT** ordered (at least, not until Python 3.6)
- Only one value per key at a time
- Error if you try to read a non-existent key

```
In [55]: empty_dict = {}
         a_dict = {'a': 1, 'b': 2, 'c': 3}
         b_dict = dict([('a', 1), ('b', 2), ('c', 3)])
         print a_dict
         print b_dict
         print a_dict['b']
         a_dict['b'] = 4
         print a_dict
         print empty_dict['a']
```

```
{'a': 1, 'c': 3, 'b': 2}
{'a': 1, 'c': 3, 'b': 2}
2
{'a': 1, 'c': 3, 'b': 4}
```

```
        KeyErrorTraceback (most recent call last)

        <ipython-input-55-4f33d9a25635> in <module>()
          7 a_dict['b'] = 4
          8 print a_dict
    ----> 9 print empty_dict['a']


        KeyError: 'a'
```

2.15 Dictionary methods

- pop(key) - removes key, value and returns value
- del a_dict[key] - removes key, value
- get(key, default) - returns value for key, or default value if key not found
- setdefault(key, default) - returns value for key, or default value if key not found and sets value for key to be default

- has_item(key), key in a_dict - returns true if key is present
- keys(), values(), items() - returns list of keys, values, or key,value pairs

```
In [59]: x = a_dict.pop('a')
         print x, a_dict

         print a_dict.get('d', None)
         print a_dict.setdefault('a', 1)
         print a_dict

         for key in a_dict.keys():  # (or) for key in a_dict:
             print key

         for key, value in a_dict.items():  # (or) for key in a_dict:
             print key, value
```

```
1 {'c': 3, 'b': 4}
None
1
{'a': 1, 'c': 3, 'b': 4}
a
c
b
a 1
c 3
b 4
```

3 Intro to Python

3.1 System Administrators Welcome

3.1.1 Day 2, Morning session

Naomi Ceder, @naomiceder

3.2 Review and questions

- Data types and variables
- Previous exercises
- Other questions?

3.3 Functions

- re-usable chunk of code, a new command

- see built-in functions

- use of `pass`

- format:

```
def function_name([parameters]):
"""docstring"""
function code block
return [value(s)]
```

```
In [21]: def greeting():
             """ prints a greeting"""
             print "hello"

         greeting()
```

```
hello
```

3.4 Function parameters

- positional
- named
- default values

```
In [22]: def add_print(one, two):
             print one + two

         add_print(1, 2)

         add_print(two=2, one=1)

         add_print(2, one=1)
```

```
3
3
```

```
        TypeErrorTraceback (most recent call last)

        <ipython-input-22-56a3ba8d99a9> in <module>()
          6 test(two=2, one=1)
          7
    ----> 8 test(2, one=1)


        TypeError: test() got multiple values for keyword argument 'one'
```

3.5 Function return values

- no return or empty return - `None`
- single return value
- multiple return values - tuple packing/unpacking

```
In [28]: def add_print(one, two):
             print one + two

         def add(one, two):
             return one + two

         def reverse(one, two):
             return two, one

         result = add_print(1, 2)
         print "add_print result:", result

         result = add(1, 2)
         print "add result", result

         result = reverse(1, 2)
         print "reverse result", result

         result_1, result_2 = reverse(1, 2)
         print "reverse result unpacked -", "result_1:", result_1, "result_2:", result_2
```

```
3
add_print result: None
add result 3
reverse result (2, 1)
reverse result unpacked - result_1: 2 result_2: 1
```

3.6 Standard program expanded

It's good practice to put your script code in separate function, and keep the `__main__` section as concise as possible.

```
In [ ]: #!/bin/env python
        """
            This is a docstring for the entire program

            You can put details about the purpose, usage, parameters,
            date, author, license, etc here.

            In general, you should use docstrings as documenation,
            and comments should be used to explain tricky bits of code
```

```
"""

def main():
    """This is the docstring for the main function
       typically it might call other functions
    """

    print "STOP! Who would cross the Bridge of Death must answer me these questions thre

    name = raw_input("What is your name? ")
    quest = raw_input("What is your quest? ")
    color = raw_input("What is your favorite color? ")
    if color == "blue":
        print "You may pass!"
    else:
        print "Arrrghhhh!!!"


# this is the "dunder" main part
if __name__ == "__main__":
    """ This is a docstring for the main part.
        Not much need in a simple program, but very handy with more complex scripts"""
    main()
```

3.7 Exercises

The unix `wc` utility takes either a file name or a stream and returns the number of lines, words, and characters (bytes) in that file.

3.8 String methods

- lower(), upper(), title()
- find(), startswith(), endswith(), in, replace(target, replacement) - find sub string in string, at beginning, end, anywhere, replace target with replacment
- strip() - removes whitespace or optional specified characters
- join(sequence) - uses the string as the "glue" to connect a sequence of strings
- split(), splitlines() - splits on white space or newlines

```
In [82]: x = "This is a TEST String"
         print x.lower(), x.upper(), x.title()

         print x.replace('TEST', 'wonderful')

         print x.find("is")
         print "is" in x
         print "   x    ".strip()
         joined = "|".join(['1', '2', '3', '4', '5'])
         print joined
```

```
        splits = joined.split("|")
        print splits
        print x.split()   # split by default on whitespace
```

```
this is a test string THIS IS A TEST STRING This Is A Test String
This is a wonderful String
2
True
x
1|2|3|4|5
['1', '2', '3', '4', '5']
['This', 'is', 'a', 'TEST', 'String']
```

3.9 String formatting

- using %
- format - %d %s %f
- %(mapping key)s

```
In [47]: print "this is a string - %s" % "test string"
         print "this is a string padded to 30 - %30s" % "test string"
         print "this is an int - %d" % 10
         print "this is a float %f" % 1.01
         print "this is a float padded to 10 total %10.3f" % 1.01
         print "this is a float 0 padded to 10 total %010.3f" % 1.01

         print "using a dict for name - %(name)s and age - %(age)d" % {'name': 'Naomi', 'age': 3
```

```
this is a string - test string
this is a string padded to 30 -                    test string
this is an int - 10
this is a float 1.010000
this is a float padded to 10 total      1.010
this is a float 0 padded to 10 total 000001.010
using a dict for name - Naomi and age - 39
```

3.10 Standard Library - useful modules

- “batteries included”
- sys
- argv - contains command line arguments, sys.argv[0] is the program name as called
- exit - exits, returning an int error code
- stdout, stdin, stderr - file-like objects for standard IO… don’t need to be opened or closed.
- random - generation of random integers, floats, random selections from a list, etc.

3.11 Command Line Options

- sys.argv
- simplest, no configuration, handy for e.g., filenames
- not flexible, no help, cumbersome for options
- argparse
- full options, help,
- requires configuration

In [43]:
```
"""
    copies from one txt file to another

"""

import sys

def main(file1, file2):
    outfile = open(file2, "w")
    with open(file1) as infile:
        for line in infile:
            outfile.write(line)


if __name__ == '__main__':
    main(sys.argv[1], sys.argv[2])
```

In [ ]:
```
"""
    copies from one txt file to another

"""

import argparse

def main(file1, file2):
    outfile = open(file2, "w")
    with open(file1) as infile:
        for line in infile:
            outfile.write(line)


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Copy a file to another')
    parser.add_argument('source',
                        help='source file')
    parser.add_argument('destination',
                         help='destination file')
    args = parser.parse_args()

    main(args.source, args.destination)
```

3.12 List comprehensions

- Alternative to a "filter" loop
- get

  ```
  new_list = [list_element for list_element in sequence (optional) if condition]
  ```

  equivalent to:

```
new_list = []
for list_element in squence:
    if condition:
        new_list.append(list_element
```

```
In [32]: # create a list of squares
         sq_list = [x * x for x in range(10)]
         print sq_list

         # create a list of even squares
         sq_list = [x * x for x in range(10) if not x % 2]
         print sq_list

         # create a list of roots, squares and cubes
         sq_list = [(x, x * x, x ** 3) for x in range(10)]
         print sq_list
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 4, 16, 36, 64]
[(0, 0, 0), (1, 1, 1), (2, 4, 8), (3, 9, 27), (4, 16, 64), (5, 25, 125), (6, 36, 216), (7, 49, 3
```

3.13 Exceptions

- way of handling errors with the code
- try, except, finally
- EAFP vs LBYL
- "duck typing"

If an error is not handled where it occurs it is passed up to the caller.

```
try:
    things that might raise an exeption
except Exception, e:
    actions to handle exception, which is e
finally:
    things that should be done in either case
```

```
In [37]: y = 0
         z = 10

         #x = z / y
         try:
             x = z / y
         except Exception, e:
             print e
         finally:
             print "always done"

integer division or modulo by zero
always done
```

3.14 Trap specific errors

Best practice is to handle specific errors and let the rest propagate back up to the stack.

Avoid a bare `except`

```
In [40]: y = 0
         z = 10
         try:
             x = z / y
         except ZeroDivisionError, e:
             z = None
             print e

         print z

         # NOT this - all errors will be treated as ZeroDivision

         try:
             x = z / y
         except:
             z = None

         print z

integer division or modulo by zero
None
None
```

3.15 Exception types

You can trap exceptions by specific type, use more than one except.

Common exceptions are:

```
ZeroDivisionError - divide by 0
IOError - e.g. file not found, no access
ImportError - module not found or importable
IndexError - list index not found
KeyError - dictionary key doesn't exist
SyntaxError - failure to close parens, no colon, etc
IndentationError - indentation not right
TypeError - type used incorrectly, e.g, adding int and string
```

https://docs.python.org/2.7/library/exceptions.html#exception-hierarchy

```
In [5]: import asdfasdf
```

```
        ImportErrorTraceback (most recent call last)

        <ipython-input-5-ce3637bca009> in <module>()
    ----> 1 import asdfasdf


        ImportError: No module named asdfasdf
```

```
In [9]: x = {1:1}
        x['a']
```

```
        KeyErrorTraceback (most recent call last)

        <ipython-input-9-20e96a6d06b8> in <module>()
          1 x = {1:1}
    ----> 2 x['a']


        KeyError: 'a'
```

```
In [7]: y = [1, 2]
        y[3]
```

```
        IndexErrorTraceback (most recent call last)
        <ipython-input-7-e2b0d37c2fde> in <module>()
          1 y = [1, 2]
    ----> 2 y[3]


        IndexError: list index out of range
```

```
In [8]: if x == 0:
        print "zero"
```

```
          File "<ipython-input-8-cfc48ccb66fd>", line 2
        print "zero"
            ^
    IndentationError: expected an indented block
```

```
In [6]: 1/0
```

```
        ZeroDivisionErrorTraceback (most recent call last)

        <ipython-input-6-05c9758a9c21> in <module>()
    ----> 1 1/0


        ZeroDivisionError: integer division or modulo by zero
```

3.16 Raising exceptions, custom exceptions

You can raise an exception with your own message

```
raise Exception("negative numbers not allowed"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
Exception: negative numbers not allowed
```

(You can also sub-classes exceptions to create custom exceptions)

3.17 Reading & Writing files

- file objects
- created using open() function
- modes

- read = "r" (or nothing)
- write = "w"
- append = "a"
- text = (nothing)
- binary = "b" (writes/reads line endings literally
- universal newline = "U"
- E.g.
- open("afile") - read a text file
- open("afile", "ab") - append to binary file
- open("afile", "rU") - read a text file with universal newlines
- open("afile", "w") - write a text file

3.18 file methods

- read()
- readlines() *(with newlines)*
- xreadlines() *(with newlines)*
- readline() *(with newline)*
- write(data)
- seek(pos) moves cursor in file to pos (as int)
- tell() gets current cursor position (as int)

```
In [8]: data = """This is
        some data
        in 3 lines"""

        # open a file for writing
        out_file = open("test", "w")

        # write data
        out_file.write(data)

        # close file
        out_file.close

        # read whole file
        in_file = open("test")
        x = in_file.read()
        print x

        # read lines
        in_file = open("test")
        x = in_file.readlines()
        print x

        # read 1 line
        in_file = open("test")
        x = in_file.readline()
        print x
        print in_file.tell()
        in_file.seek(14)
        x = in_file.readline()
        print x

This is
some data
in 3 lines
['This is \n', 'some data\n', 'in 3 lines']
This is

9
data
```

3.19 Files as sequence of lines

A file can also be treated as a sequence of lines:

```
for line in open("test"):
    print line
```

But note that new lines will not be stripped (just like readlines())

```
In [9]: for line in open("test"):
            print line

This is

some data

in 3 lines
```

3.20 Easier error handling with contexts and `with`

Contexts are a way to wrap operations like file I/O with standard entry and exit procedures. In the case of files, it ensures that the file is closed promtly after the operation.

```
In [12]: with open("test") as in_file:
             for line in in_file:
                 print line
         # file is closed

This is

some data
```

```
in 3 lines
```

```
In [ ]:
```

3.21 Exercises

Write a program to find out who is hogging disk space

```
In [ ]:
```

3.22 Intro to Python

3.23 System Administrators Welcome

3.23.1 Day 2, Afternoon session

Naomi Ceder, @naomiceder

3.24 More about modules

When you load module:

1. the code is located loaded (executed) - ImportError if it can't be found/imported
2. the module's namespace is bound to it's name in the loading module's namespace - `import x`, `import x as y`
3. OR the specifically imported names are copied into the loading modules namespace - `from x import z`

3.25 sys.path

- a list of directories Python searches in order for modules/packages
- set by PYTHONPATH env variable at startup or programmatically

3.26 sys.modules

- dictionary of imported modules
- if module is in sys.modules, it's immediately used
- code is only executed the first time it's imported
- module's namespace is accessible to importing module

3.27 if not in sys.modules

- interpreter locates module's .py file
- looks for compiled .pyc file
- if .pyc if found, it's used if newer
- else module is recompiled to create new

```
In [ ]:
```

3.28 Data files - CSV

- reads and writes almost any delimited file, not just comma delimited
- takes care of quoting as needed

```
In [24]: import csv
         data = [['cust_number', 'city, state', 'balance'],
                 [1234, 'Chicago, IL', 100.00],
                 [1235, 'New York, NY', 10.00]]

         csv.writer(open("test.csv", "w")).writerows(data)
         print open("test.csv").read()

         reader = csv.reader(open("test.csv"))
         for row in reader:
             print row
```

```
cust_number,"city, state",balance
1234,"Chicago, IL",100.0
1235,"New York, NY",10.0

['cust_number', 'city, state', 'balance']
['1234', 'Chicago, IL', '100.0']
['1235', 'New York, NY', '10.0']
```

3.29 Data files - JSON

- parse a string or file into JSON object
- write JSON object to a string or file

```
In [62]: import json
         data = [{'cust_number': 1234, 'city, state': 'Chicago, IL', 'balance': 100.00},
                 {'cust_number': 1235, 'city, state': 'New York, NY', 'balance': 10.00}]
         data_json =  json.dumps(data, indent=2)
         print data_json
         data_2 = json.loads(data_json)
         print data_2
```

```
[
  {
    "balance": 100.0,
    "cust_number": 1234,
    "city, state": "Chicago, IL"
  },
  {
    "balance": 10.0,
    "cust_number": 1235,
    "city, state": "New York, NY"
  }
]
[{u'balance': 100.0, u'cust_number': 1234, u'city, state': u'Chicago, IL'}, {u'balance': 10.0, u
```

3.30 Regular expressions

- A parsing/pattern matching mini language for string searches & replacements
- Powerful, but tricky

"Some people, when confronted with a problem, think "I know, I'll use regular expressions." Now they have two problems." Jamie Zawiski

3.31 Caveats

Don't use regex unless you need to. You might be just as happy using something simpler:

- finding a string = s.find(target), s.startswith(target), s.endswith(target), target in s (handle uppler/lower cases with lower())
- replacing a string - new_string = s.replace(target, replacement)

3.31.1 Example - Phone numbers

US phone numbers are 10 digits long, formatted in various ways - the first 3 digits in parentheses, then 3 digits, a dash, then 4 digits; 3 digits, dash, 3 digits, dash, 4 digits; the same 3 3 4 grouping separated by spaces; sometimes the same 3 3 4 grouping separated by periods; or sometimes with no separators at all.

These are all possible phone number formattings:

- (123) 456-7890
- 123-456-7890
- 123 456 7890
- 123.456.7890
- 1234567890

```
In [10]: import re

         data = """123-45-6789
         234 567 1234
         (900)123-1234
         900-100-5000
         abc-123-1234
         198765432"""
         lines = data.split("\n")

         for line in lines:
            result = re.match(r"\(*\d{3}\)*[ -]*\d{3}[ -]*\d{4}", line)
            if result:
                print result.group(), "is a phone number"
            else:
                print line, "is NOT a phone number"
```

```
123-45-6789 is NOT a phone number
234 567 1234 is a phone number
(900)123-1234 is a phone number
900-100-5000 is a phone number
abc-123-1234 is NOT a phone number
198765432 is NOT a phone number
```

3.32 System commands

- os.system(), os.spawn(), os.popen() **NOT RECOMMENDED** - executes a command on the system
- popen2 **DEPRECATED** - creates subprocesses and pipes inputs and outputs
- subprocess - recommended for spawning new processes, connect to their input/output/error pipes, and obtain their return codes

3.33 subprocess module

- call() - easy execution of a process
- check_outpue() - easy call with retreival of output
- Popen - class for piping together several processes

```
In [47]: import subprocess
         output = subprocess.check_output(["ls", "-ltr"])
         print output

         proc = subprocess.Popen(["ls", "-ltr"], stdout=subprocess.PIPE)
         stdout, stderr = proc.communicate()
         print stdout
         print stderr
```

```
total 0
lrwxrwxrwx 1 nbuser nbuser 28 Mar  3 23:04 anaconda2_410 -> /home/nbcommon/anaconda2_410
lrwxrwxrwx 1 nbuser nbuser 28 Mar  3 23:04 anaconda3_410 -> /home/nbcommon/anaconda3_410

total 0
lrwxrwxrwx 1 nbuser nbuser 28 Mar  3 23:04 anaconda2_410 -> /home/nbcommon/anaconda2_410
lrwxrwxrwx 1 nbuser nbuser 28 Mar  3 23:04 anaconda3_410 -> /home/nbcommon/anaconda3_410

None
```

3.34 OS library - Operating system interfaces

The os library has a number of services for interacting with the operating system, in particular process information and control and file information and management.

os module documentation

- os.environ - reads environment variables
- os.mkdir - creates director
- os.rename - renames/moves file
- os.remove - removes file
- os.walk - traverses file tree
- os.kill - kills process

```
In [ ]: # from the Python documentation... https://docs.python.org/2.7/library/os.htm

        # Delete everything reachable from the directory named in "top",
        # assuming there are no symbolic links.
        # CAUTION:  This is dangerous!  For example, if top == '/', it
        # could delete all your disk files.
        import os
        for root, dirs, files in os.walk(top, topdown=False):
            for name in files:
                os.remove(os.path.join(root, name))
            for name in dirs:
                os.rmdir(os.path.join(root, name))
```

3.35 os.path

There is also a submodule `os.path` which offers numerous operations on file paths. os.path documentation

- join() - combines list of elements into path using correct path separator for the OS
- split() - splits a path on the OS path separator
- abspath() - reports if a path is absolute from root of filesystem
- exists() - reports if a path exists on the filesystem
- isdir() - reports if path refers to a directory

```
In [5]: import os.path
        new_path = os.path.join("/home", "naomi")
        print new_path
        print os.path.split(new_path)
        print os.path.exists(new_path)
```

```
/home/naomi
('/home', 'naomi')
False
```

3.36 Fetching files over the network

- Use subprocess to control system utilities like wget, curl, etc.
- Use the ftp module to access ftp servers
- Use the requests module to access HTTP/HTTPS services

3.37 Resources

- Python Documentation
- Standard Library
- Tutorial
- Elements of Python Style
- Hitchhikers Guide to Python

3.38 Tour of Python Standard library

Index of standard library (sleep with this under your pillow)

```
In [ ]:
```