

# SI 506: Programming I

## Fall 2019

### Lecture 03

Anthony Whyte <[arwhyte@umich.edu](mailto:arwhyte@umich.edu)>  
Lecturer III, School of Information  
715 N. University Ave, Ann Arbor, MI 48109  
Roumanis Square, 2nd floor (“the loft”)



# preliminaries

# Top 20 programming languages

## IEEE Spectrum 2019 survey: Python ranked no. 1

Institute of Electrical and Electronics Engineers

Rank	Language	Type	Score
1	Python	🌐 🖥️⚙️	100.0
2	Java	🌐 📱 🖥️	96.3
3	C	📱 🖥️⚙️	94.4
4	C++	📱 🖥️⚙️	87.5
5	R	🖥️	81.5
6	JavaScript	🌐	79.4
7	C#	🌐 📱 🖥️⚙️	74.5
8	Matlab	🖥️	70.6
9	Swift	📱 🖥️	69.1
10	Go	🌐 🖥️	68.0

“Python’s popularity is driven in no small part by the vast number of specialized libraries available for it, particularly in the domain of artificial intelligence, where the Keras library is a heavyweight among deep-learning developers: Keras provides an interface to the TensorFlow, CNTK, and Theano deep-learning frameworks and tool kits. Deep learning isn’t the only field where Python is having an impact that could not have been anticipated when the language was first released in 1991. The dramatic increase in computing power found in microcontrollers means that embedded versions of Python, such as CircuitPython and MicroPython, are becoming increasingly popular among makers.”

Source: <https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019>



# Readings

## Available on Saturday per request

The screenshot shows a course management system interface. On the left is a dark sidebar with various icons and links. The main area has a header 'SI 506 001 FA 2019 > Assignments > Readings'. The content area is titled 'Readings' and shows a section for 'Week 02 Recommended Readings'. It includes a 'Published' button, an 'Edit' button, and a 'Related Items' section with a 'SpeedGrader™' link. The main content lists several recommended readings:

- Charles Severance,** [Python for Everybody. Exploring data in Python 3](#) (CreateSpace, 2016).
  - Read Chapter 2, "Variables, Expressions, and Statements"
  - Our own Dr Chuck's take on the vocabulary and concepts explored in the Ceder and Bell readings.
  - Note: EPUB and PDF versions are stored locally in [Files](#). Chuck's book has been translated into several languages and is available in various formats (i.e., HTML, PDF, EPUB, Mobi) at <https://www.py4e.com/book>.
- Naomi Ceder,** [The Quick Python Book, Third Edition](#) (Manning Publications, 2018).
  - Read Chapter 4. "[The absolute basics](#)."
  - I like the pace that Naomi sets in the chapter (no fuss, straight to the point).
- Jake VanderPlas,** [A Whirlwind Tour of Python](#) (O'Reilly Media, 2016).
  - Read section "[Basic Python Semantics: Variables and Objects](#)".
  - A short section that should help clarify what constitutes an object in Python (in short, everything) and how to think about variables. The excerpt is also available in [Github](#).
- Ana Bell,** [Get Programming: Learn to code with Python](#) (Manning Publications, 2018).
  - Read Lesson 4. "[Variables and expressions: giving names and values to things](#)"
  - Read Lesson 5. "[Object types and statements of code](#)"
  - These two lessons provide a solid base for coming to grips with this week's lectures and lab.
- Lisa Tagliaferri,** "[An Introduction to String Functions in Python 3](#)" (DigitalOcean, Nov 2016).
  - A primer on using built-in string functions like `.join()`, `.split()`, and `.replace()` plus the always useful `len()` function which returns the number of elements in an object.



# O'Reilly playlist per request

<http://bit.ly/2IDVxYI>

The screenshot shows the O'Reilly Learning Platform interface. On the left is a sidebar with navigation links: Browse, Resource Centers, Playlists (selected), History, Topics, Learning Paths, Conferences, Newsletters, Highlights, Settings, Support, and Sign Out. The main area displays a playlist titled "SI 506-2019Fall" with 3 items. The first item is "A Whirlwind Tour of Python" by Jake VanderPlas, the second is "Get Programming: Learn to code with Python" by Ana Bell, and the third is "The Quick Python Book, Third Edition" by Naomi Ceder. Each item has a "BOOK" icon, a thumbnail, the title, the author, and three small icons for edit, copy, and share.

OREILLY®

Find a Solution...

Filter by: All

Search

Browse

Resource Centers

Playlists

History

Topics

Learning Paths

Conferences

Newsletters

Highlights

Settings

Support

Sign Out

19 items

SI 506

45 items

SI 506-2019Fall

3 items

SIADS 501

2 items

DESCRIPTION

SI 506 (2019 Fall) recommended readings available on the O'Reilly Learning Platform.

Infinispan Python

PUBLIC 3 FOLLOWERS

Last Updated: Sep 7, 2019

A Whirlwind Tour of Python

By Jake VanderPlas

Get Programming: Learn to code with Python

By Ana Bell

The Quick Python Book, Third Edition

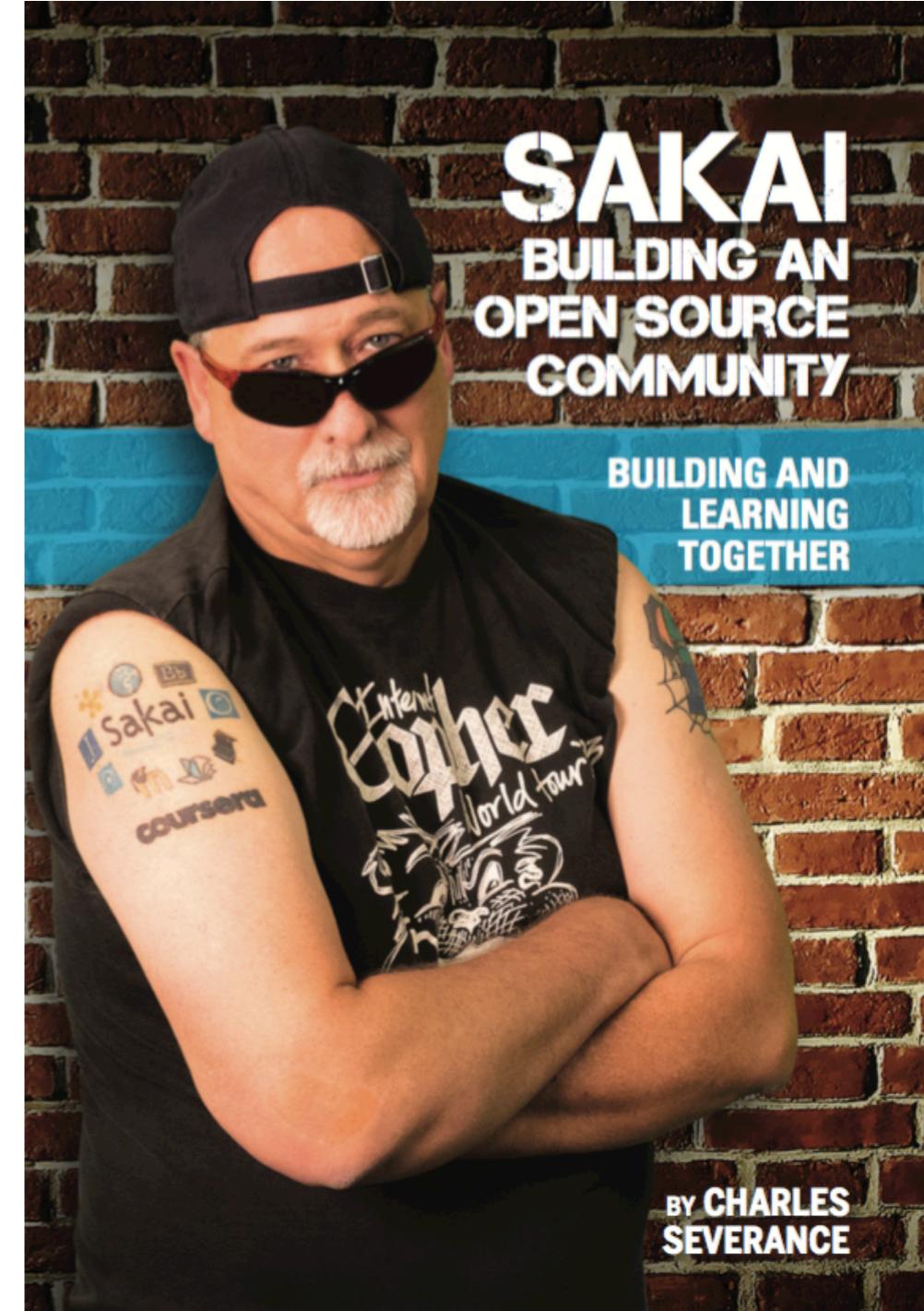
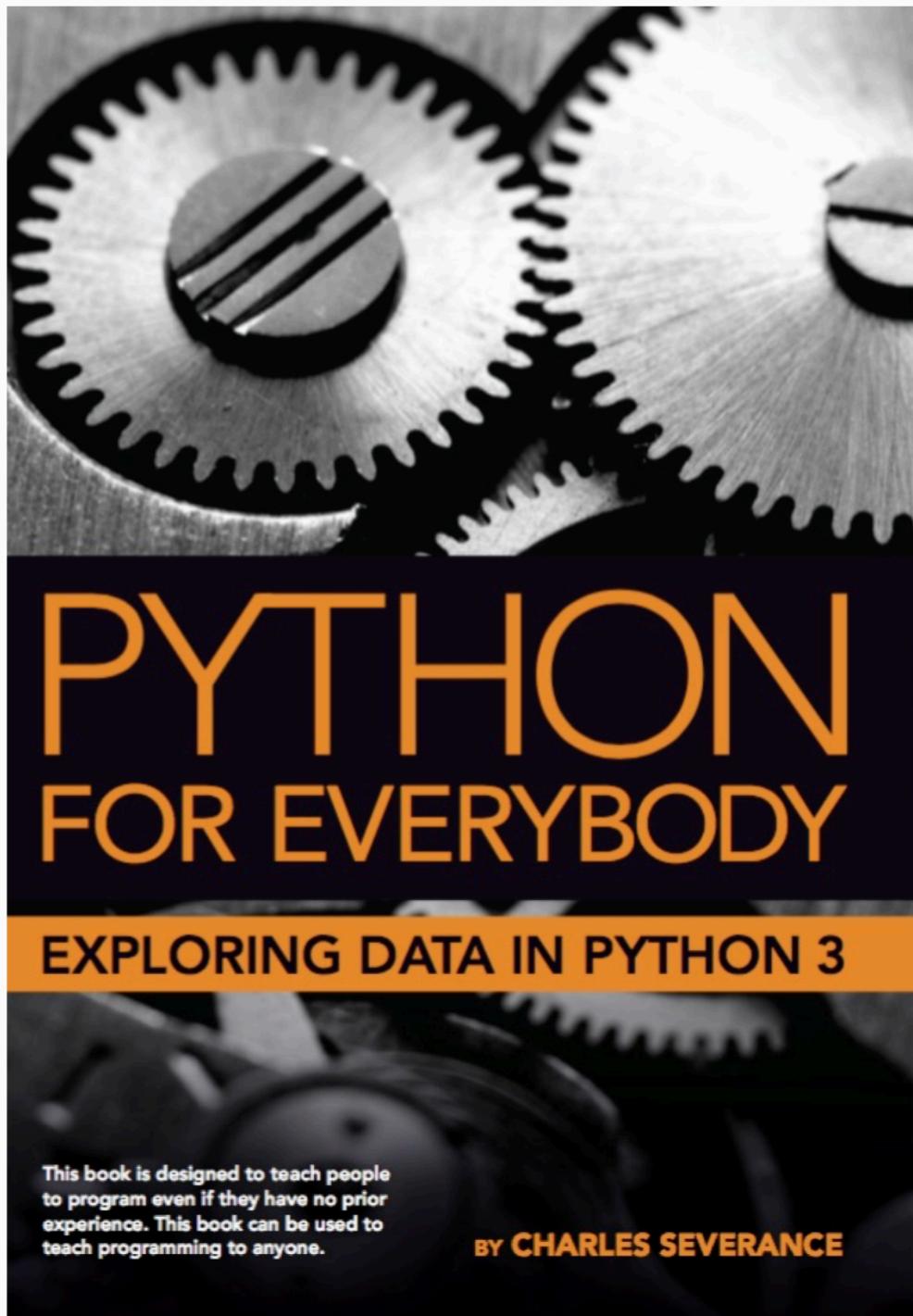
By Naomi Ceder

<https://learning.oreilly.com/playlists/d48870e7-033e-48b8-907d-1a56868bb089>



# Readings: Dr Chuck

If new to programming, start with P4E chapters



# assignments



# Problem set 01/Lab exercise 02

## design, workflow

### SI 506: Problem Set 01

This week's problem set includes six (6) one to two line problems that focus on variables creation and value assignment, string concatenation, use of arithmetic operators, and use of certain built-in functions discussed in the lectures and the readings.

Retrieve the problem set 01 template file at [https://github.com/umsi-arwhyte/SI506-2019Fall/code/gradescope/problem\\_set\\_01.py](https://github.com/umsi-arwhyte/SI506-2019Fall/code/gradescope/problem_set_01.py)

When you have completed the problem set click on the Gradescope link in Canvas and upload your `problem_set_01.py` file to the Gradescope site. Your submission will be auto-graded and any runtime errors encountered will be recoded and displayed. You may re-submit your assignment as many times as is necessary before the close date. Late submissions will be penalized as described in the syllabus.

The general layout of a SI 506 problem set resembles the following:

```
# START PROBLEM SET 01
print('PROBLEM SET 01 \n')

# PROBLEM 01A (25 points)
# Instructions for problem 01A provided here. Read carefully.

# BEGIN 01A SOLUTION

# Write your 01A solution between the BEGIN/END comments.

# Note that for certain problems a required variable may be
# initialized with a default value (e.g., var_int = 0, var_str = '')
# that you will then change. print() statements are occasionally
# provided as a courtesy in order to output variable values
# to the screen.

# END 01A SOLUTION

# START PROBLEM 01B SETUP (do not modify)

# WARNING: For some problems a "setup" section is included in order
# to provide variables, functions or other working code for use
# with the problem(s). Do not change any code located between
# START/END SETUP.

# END SETUP

# PROBLEM 01B (25 points)
# Instructions for problem 01B provided here. Read carefully.

# BEGIN 01B SOLUTION

# Write your 01B solution between the BEGIN/END comments. . .

# END 01B SOLUTION

# Additional problems will follow, structured similarly to problems 01A and 01B above.

# END PROBLEM SET
```

### SI 506: Lab Exercise 02

This week's lab exercise includes two (2) one to two line problems that focus on variables creation and value assignment, string concatenation, and use a certain built-in function discussed in the lectures and the readings.

Retrieve the lab exercise 02 template file at [https://github.com/umsi-arwhyte/SI506-2019Fall/code/gradescope/lab\\_exercise\\_02.py](https://github.com/umsi-arwhyte/SI506-2019Fall/code/gradescope/lab_exercise_02.py)

When you have completed the problem set click on the Gradescope link in Canvas and upload your `lab_exercise_02.py` file to the Gradescope site. Your submission will be auto-graded and any runtime errors encountered will be recoded and displayed. You may re-submit your exercise as many times as is necessary before the close date. Late submissions will be penalized as described in the syllabus.

The general layout of a SI 506 lab exercise resembles the following:

```
# START LAB EXERCISE 01
print('LAB EXERCISE 01 \n')

# PROBLEM 01A (25 points)
# Instructions for problem 01A provided here. Read carefully.

# BEGIN 01A SOLUTION

# Write your 01A solution between the BEGIN/END comments.

# Note that for certain problems a required variable may be
# initialized with a default value (e.g., var_int = 0, var_str = '')
# that you will then change. print() statements are occasionally
# provided as a courtesy in order to output variable values
# to the screen.

# END 01A SOLUTION

# START PROBLEM 01B SETUP (do not modify)

# WARNING: For some problems a "setup" section is included in order
# to provide variables, functions or other working code for use
# with the problem(s). Do not change any code located between
# START/END SETUP.

# END SETUP

# PROBLEM 01B (25 points)
# Instructions for problem 01B provided here. Read carefully.

# BEGIN 01B SOLUTION

# Write your 01B solution between the BEGIN/END comments. . .

# END 01B SOLUTION

# Additional problems may follow, structured similarly to problems 01A and 01B above.

# END LAB EXERCISE
```

# Lab: Gradescope workflow

## auto grading with Python

gradescope 

< SI 506 2019 Fall

**Problem Set 000**

- Configure Autograder
- Manage Submissions
- Review Grades

---

-  Regrade Requests
-  Statistics
-  Settings

### Autograder Results

Results 

Autograder Output (hidden from students)

PROBLEM SET 000

```
jabberwocky='Twas brillig, and the slithy toves  
Did gyre and gimble in the wabe;  
All mimsy were the borogoves,  
And the mome raths outgrabe.  
  
"Beware the Jabberwock, my son!  
The jaws that bite, the claws that catch!  
Beware the Jubjub bird, and shun  
The frumious Bandersnatch!"  
  
He took his vorpal sword in hand:  
Long time the manxome foe he sought—  
So rested he by the Tumtum tree,  
And stood awhile in thought.  
  
And as in uffish thought he stood,  
The Jabberwock, with eyes of flame,  
Came whiffling through the tulgey wood,  
And burbled as it came!  
  
One, two! One, two! And through and through  
The vorpal blade went snicker-snack!  
He left it dead, and with its head  
He went galumphing back.  
  
"And hast thou slain the Jabberwock?  
Come to my arms, my beamish boy!  
O frabjous day! Callooh! Callay!"  
He chortled in his joy.  
  
'Twas brillig, and the slithy toves  
Did gyre and gimble in the wabe;  
All mimsy were the borogoves,  
And the mome raths outgrabe.
```

num\_chars=933

num\_words=166

Evaluate character count for equality.  
(100.0/100.0)

Evaluate word count for equality.  
(100.0/100.0)

STUDENT  
Anthony Whyte

AUTOGRADE SCORE  
200.0 / 200.0

PASSED TESTS

Evaluate character count for equality. (100.0/100.0)  
Evaluate word count for equality. (100.0/100.0)

Account 

More 

> Debug via SSH

Submission History 

Download Submission 

Resubmit 

# back to Python



# Python objects

## object thinking

“everything is an object”

Jake VanderPlas, *A Whirlwind Tour of Python*



# Python variables

not a container; think name or label

“The name *variable* is somewhat misleading . . . ; *name* or *label* would be more accurate.”

Naomi Ceder, *The Quick Python Book*, chapter 4

# Python variables

the good doctor agrees

**“A variable is a name that refers to a value.”**

Charles Severance Python for Everybody, chapter 2

# Python variables

Jake likes the notion of a ‘pointer’

“. . . variables are simply pointers [to objects], and the variable names themselves have no attached type information.”

*Jake VanderPlas, A Whirlwind Tour of Python*

# Test the assertions

Start your Python console

[Send feedback](#) [Forums](#) [Help](#) [Blog](#) [Account](#) [Log out](#)



[Dashboard](#) [Consoles](#) [Files](#) [Web](#) [Tasks](#) [Databases](#)

## Dashboard

Welcome, [nantin](#)

**CPU Usage:** 1% used – 1.13s of 100s. Resets in 22 hours, 29 minutes [More Info](#)

[Upgrade Account](#)

**File storage:** 0% full – 148.0 KB of your 512.0 MB quota

Recent  
Consoles

+ 5 -

You have no recent consoles.

New console:

\$ Bash

>> Python ▾

More...

Recent  
Files

+ 5 -

You have no recently edited files.

+ Open another file

Browse files

Recent  
Notebooks

+ 5 -

Your account does not support Jupyter Notebooks. [Upgrade your account](#) to get access!

All  
Web apps

You don't have any web apps.

[Open Web tab](#)



# Test the assertions

interactive shell (type this)

```
$ python3
>>> x = [506, 507]      >>> x = y
>>> y = x                >>> print(x)
>>> x.append(508)         >>> del x # delete
>>> print(x)              >>> print(x) # oops
>>> print(y)              >>> print(y)
>>> x = 'five oh six'
>>> print(y)
```

# Objects and variables (cont.)

name error *not* object not found error

```
$ python3
```

```
...
```

```
>>> del x
```

```
>>> print(x)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
NameError: name 'x' is not defined
```

# Observations

regarding x and y name/label/pointers

1. x and y point to the same list (object)
2. new element added to list referenced by x
3. list that y references also changes (same object, beware)
4. x assigned to new object (override previous assignment)
5. y continues to reference list
6. x (re)assigned to list referenced by y
7. x deleted
8. print(x) throws traceback (NameError)
9. y continues to reference list
10. types are associated with objects *not* variables

# built-in functions

# Function

custom code block that encapsulates a unit of work

```
def some_func(arg_01, arg_02):
    """
    docstring (describes what this function does)
    """

    # Do something useful, e.g.,

    # Accept two inputs (i.e., parameters/arguments)
    # Process inputs
    # Provide a result

    return something
```

# Built-in functions

Python Interpreter provides a set of pre-defined functions

69 built-in functions currently available

Our focus (this week)

`print()`

`type()`

`len()`

`input()`



# Built-in function: print()

print one or more objects to a stream or file-like object

`print(*objects, sep=' ',end='\n', file=sys.stdout, flush=False)`

Print `objects` to the text stream `file`, separated by `sep` and followed by `end`. `sep`, `end`, `file` and `flush`, if present, must be given as keyword arguments.

Source: <https://docs.python.org/3/library/functions.html#print>. See also <https://realpython.com/python-print/>



# Built-in function: type()

evaluate object type

## type(object)

With one argument, return the type of an object. The return value is a type object and generally the same object as returned by `object.__class__`.

Source: <https://docs.python.org/3/library/functions.html#type>



# Built-in function: `len()`

evaluate sequences and collections

## `len(object)`

Return the `length` (the number of items) of an object. The argument may be a `sequence` (such as a string, bytes, tuple, list, or range) or a `collection` (such as a dictionary, set, or frozen set).

Source: <https://docs.python.org/3/library/functions.html#len>



# Built-in function: `input()`

request/return user input

## `input([prompt])`

If the `prompt` argument is present, it is written to standard output without a trailing newline. The function then reads a line from `input`, converts it to a string (stripping a trailing newline), and returns that. When EOF is read, `EOFError` is raised.

Source: <https://docs.python.org/3/library/functions.html#input>



# Built-in functions

interactive shell (type this)

```
$ python3
>>> y = input("Type 'spam' 3 times: ")
>>> print(y)
>>> type(y)
>>> y.__class__ # just to confirm
>>> z = y
>>> print(y, z, sep=' wonderful ', end='\n\n\n')
>>> len(z)
>>> type(len(z)) # function as argument
```

# string functions



# String functions

Built-in types also provisioned with functions (very handy)

Our focus (this week)

`str.split()`

`str.replace()`

`str.join()`

# Built-in function: str.split()

return a list of character chunks per chosen delimiter

**str.split(sep=None, maxsplit=-1)**

Return a list of the words in the string, using sep as the delimiter string. If maxsplit is given, at most maxsplit splits are done (thus, the list will have at most maxsplit+1 elements). If maxsplit is not specified or -1, then there is no limit on the number of splits (all possible splits are made).

If sep is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings ....

Source: <https://docs.python.org/3/library/stdtypes.html#str.split>



# Built-in function: str.split()

interactive shell (type this)

```
$ python3
>>> s = 'spam,spam,spam'
>>> s.split()
>>> s.split(',')
>>> s.split(sep=',', maxsplit=1)
>>> s = 'spam spam spam'
>>> p = s.split()
>>> type(p)
>>> len(p)
```

# Built-in function: str.replace()

swap out one substring for another

**str.replace(old, new[, count])**

Return a copy of the string with all occurrences of **substring** old replaced by new. If the optional argument **count** is given, only the first count occurrences are replaced.

Source: <https://docs.python.org/3/library/stdtypes.html#str.replace>



# Built-in functions

interactive shell (type this)

```
$ python3
>>> e = s.replace('spam', 'egg')
>>> print(e)
>>> print(s)
>>> b = s.replace('spam', 'egg bacon +', 1)
>>> print(b)
>>> m = s[:9].replace('spam', 'egg bacon +', 1)
>>> print(m) # actually on the menu (Spam skit)
```

See: [https://en.wikipedia.org/wiki/Spam\\_\(Monty\\_Python\)](https://en.wikipedia.org/wiki/Spam_(Monty_Python))



# Built-in function: str.join()

concatenate strings with an iterable (say what?)

## str.join(iterable)

Return a string which is the **concatenation** of the strings in **iterable**. A `TypeError` will be raised if there are any non-string values in `iterable`, including `bytes` objects. The separator between elements is the string providing this method.

Source: <https://docs.python.org/3/library/stdtypes.html#str.replace>



# Iterable?: better check stackoverflow

wisdom of the crowd (with limits)

<http://bit.ly/2kCbgXL>

# Built-in function: str.join()

interactive shell (type this)

```
$ python3
>>> print(s)
>>> t = s + ' ' + s # operator
>>> print(t)
>>> t = ".join([s, ' ', s]) # join to empty string"
>>> print(t)
>>> u = ".join([t, ' baked beans ', s, ' and spam'])"
>>> print(u) # also on the menu (Spam skit)
```

See: [https://en.wikipedia.org/wiki/Spam\\_\(Monty\\_Python\)](https://en.wikipedia.org/wiki/Spam_(Monty_Python))





finis



# directors cut



# si506\_lab\_01.py

## script anatomy and workflow

load

```
import ...
```

module imports

entry point

```
main(argv)
```

process, delegate, print

```
huzzah(name)
```

delegate, build, format

```
umich_now()
```

transform

invoke

```
if __name__ == '__main__':
    main(sys.argv[1:]) # do something
```

# Python object

the stuff of an object

Identity

Type

Value

Source: <https://docs.python.org/2.0/ref/objects.html>



# Built-in Types

a.k.a integer, float, boolean, string, list, dictionary, tuple, class, etc.

- numerics
- sequences
- mappings
- classes
- instances
- exceptions

# Variable

naming do's and don'ts

## Good

```
course_short_name = 'SI506'  
course_list = ['SI506', 'SI507', 'SI508']
```

## Bad

```
c = 'SI506' # opaque (unfriendly)  
courseList = ['SI506', 'SI507', 'SI508'] # camelcase
```

## Ugly (interpreter will complain)

```
506_umsi = 'SI506'  
$number = 506  
course-list = ['SI506', 'SI507', 'SI508']  
course name = 'SI506'
```



# Keywords

reserved: cannot be used as ordinary identifiers

False            await            else            import            pass

None            break            except            in            raise

True            class            finally            is            return

and            continue            for            lambda            try

as            def            from            nonlocal            while

assert            del            global            not            with

async            elif            if            or            yield

Source: [https://docs.python.org/3/reference/lexical\\_analysis.html?highlight=reserved%20keywords#keywords](https://docs.python.org/3/reference/lexical_analysis.html?highlight=reserved%20keywords#keywords)



# Console exercise

interactive shell (type this)

```
$ python3
```

```
...
```

```
>>> class = 'SI 506'
```

# operators and operands

(a brief intro)



# Operators

## Types

- arithmetic
- comparison
- assignment
- logical (and, or, not)
- membership (in, not in)
- identity (is, is not)
- bitwise

# Arithmetc operators

operations on operands x and y

Operator	Description	Example
+	Addition	$x + y = 506$
-	Subtraction	$x - y = 494$
*	Multiplication	$x * y = 3000$
/	Division	$x / y = 83.33$
%	Modulus (divides x by y and returns remainder)	$x \% y = 2$
**	Exponent (exponential power calculation)	$x^{**}y = 1562500000000000$
//	Floor division (result rounded to whole number)	$x // y = 83$

Q: what are the values assigned to x and y?

and now for something  
completely different



# Zen of Python

type this

```
$ python3
```

```
...
```

```
>>> import this
```