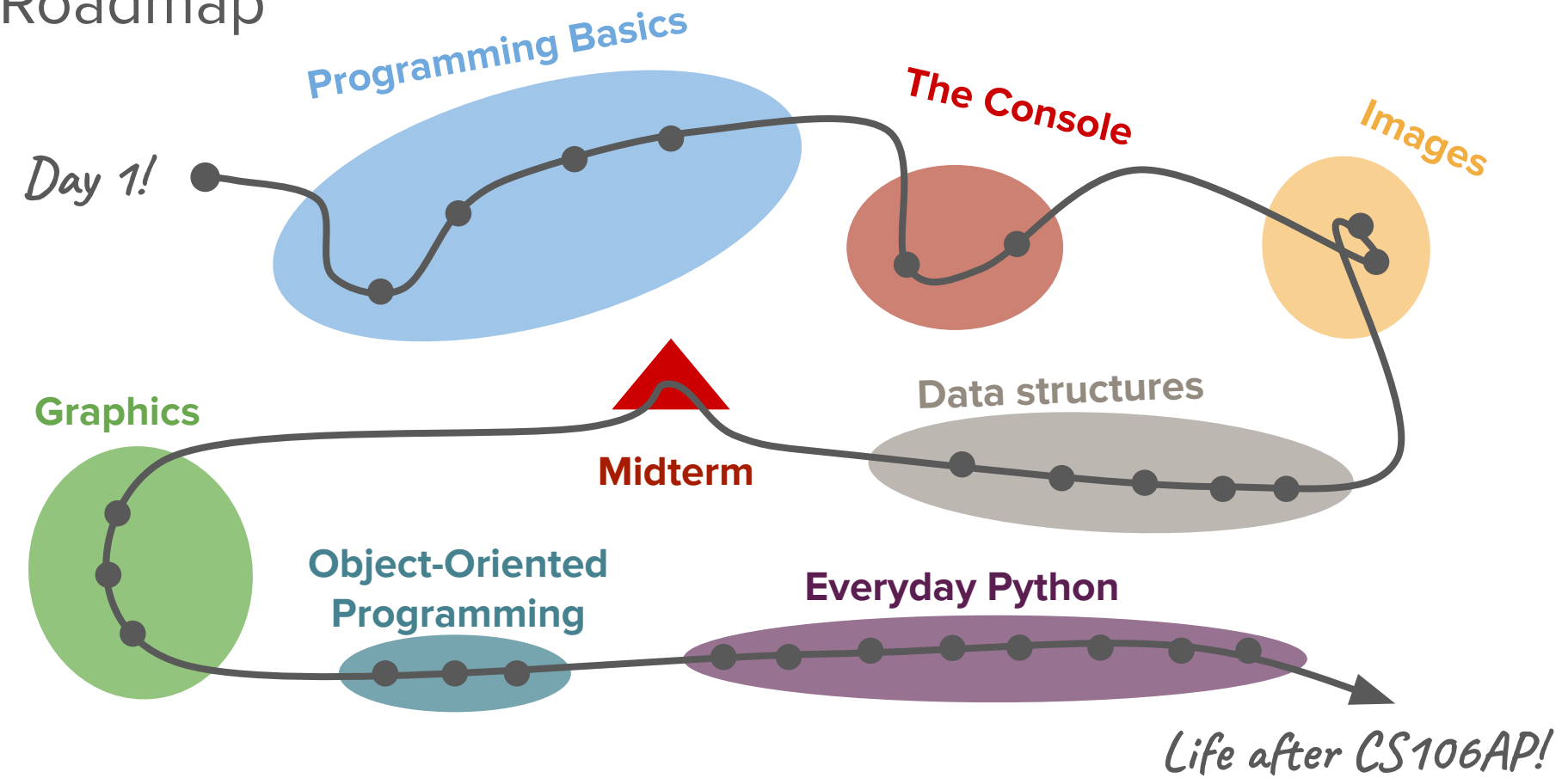
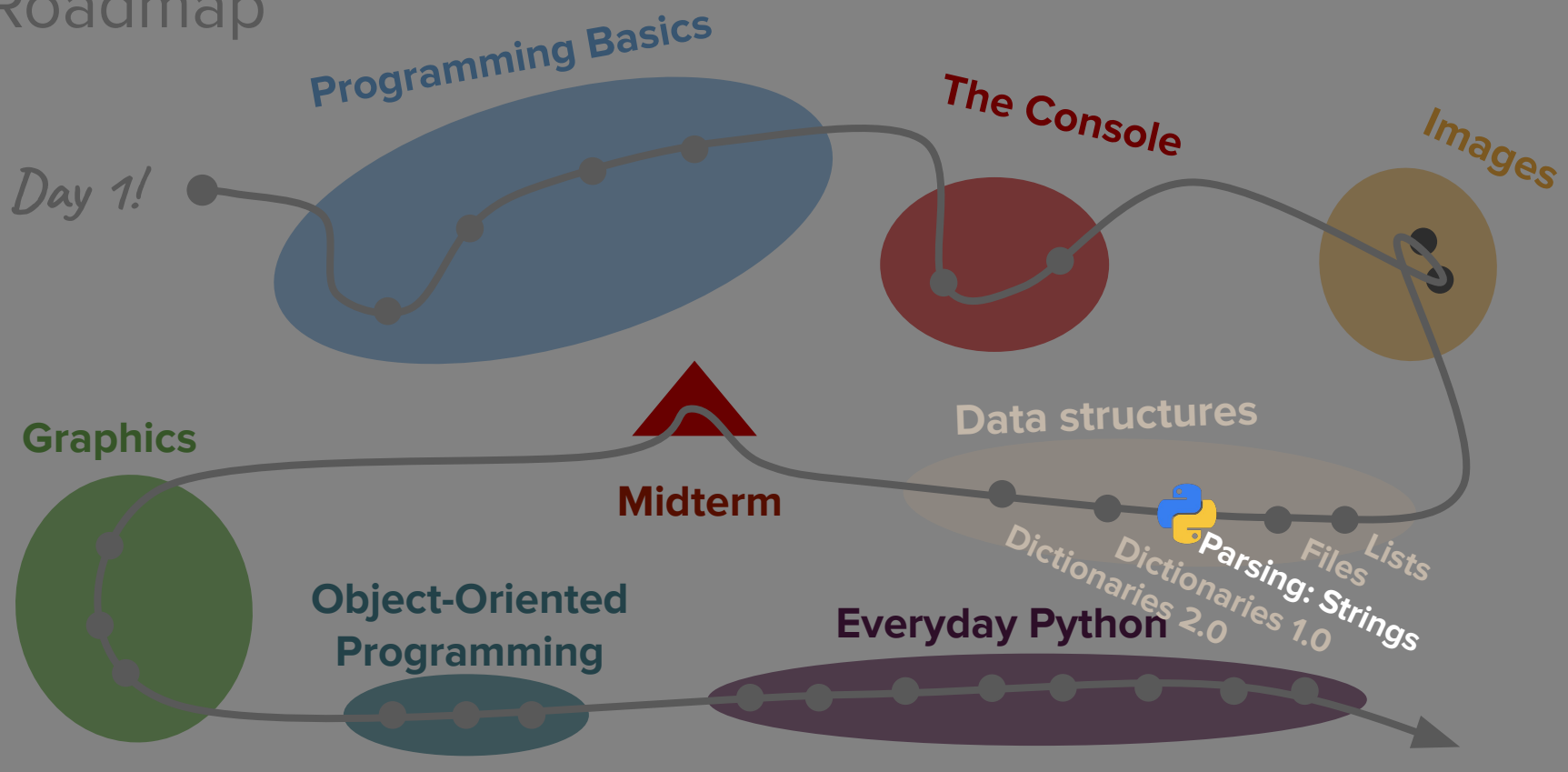
Files and Parsing

CS106AP Lecture 12

Roadmap


Programming Basics
Day 1!
The Console
Images
Graphics
Midterm
Data structures
Object-Oriented Programming
Everyday Python
Life after CS106AP!

Roadmap


Programming Basics
Day 1!
The Console
Images
Graphics
Midterm
Data structures
Parsing: Strings
Files
Lists
Dictionaries 1.0
Dictionaries 2.0
Object-Oriented Programming
Everyday Python
Life after CS106AP!

Today’s questions

How can I separate valuable data from junk?

Today's topics

1. Review
    - Command Line
    - File Reading
2. What is Parsing?
    - Useful String Functions
    - How to Parse
3. What's next?

Review

Command Line & Arguments

Command Line

PyCharm Terminal == Terminal/Command Prompt

Definition

Command Line/Terminal

Text interface for giving instructions to the computer. These instructions are relayed to the computer's operating system.

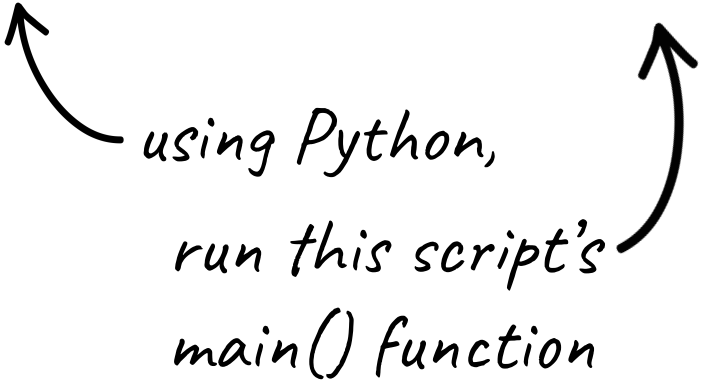
Definition

Python Console/Interpreter

An interactive program that allows us to write Python code and run it line-by-line.

Command Line Usage

```
python3 script_name.py
```



What’s up with $?

Our convention is to let "$" represent the terminal prompt.

What’s up with $?

Our convention is to let "$" represent the terminal prompt.

e.g.

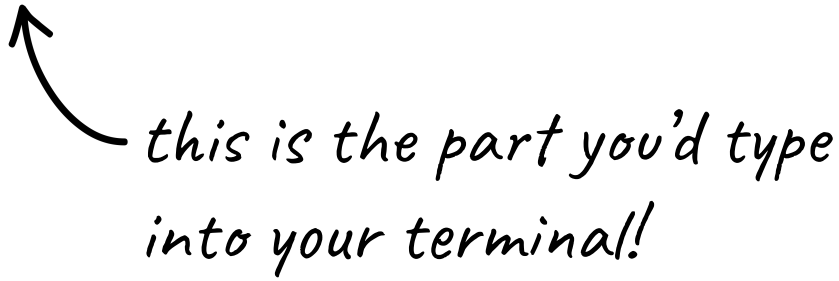
```
$ python3 ghost.py hoover
```

What's up with $?

Our convention is to let "$" represent the terminal prompt.

e.g.

```
$ python3 ghost.py hoover
```



What's up with $?

Our convention is to let "$" represent the terminal prompt.

e.g.

```
$ python3 ghost.py hoover
```

If we use ">>>", we're referring to the Python interpreter.

```
>>> 3 * 6
18
```

Think/Pair/Share:

Line-by-line: what’s happening in the following code?

Arguments

```
def main():
    args = sys.argv[1:]
    if len(args) == 1:
        print_processed_text(args[0], 'aei')
    if len(args) == 3 and args[0] == '-chars':
        print_processed_text(args[2], args[1])
```

Think/Pair/Share:
Line-by-line: what's happening in the following code?

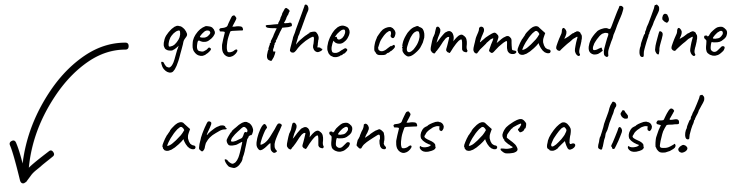
Arguments

```
def main():
    args = sys.argv[1:]
    if len(args) == 1:
        print_processed_text(args[0], 'aei')
    if len(args) == 3 and args[0] == '-chars':
        print_processed_text(args[2], args[1])
```

Think/Pair/Share:
Line-by-line: what's happening in the following code?

```
$ python3 DeleteCharacters.py -chars aei poem.txt
```

Arguments




```
def main():
    args = sys.argv[1:]
    if len(args) == 1:
        print_processed_text(args[0], 'aei')
    if len(args) == 3 and args[0] == '-chars':
        print_processed_text(args[2], args[1])
```

```
$ python3 DeleteCharacters.py -chars aei poem.txt
```

Arguments

```
def main():
    args = sys.argv[1:]
    if len(args) == 1:
        print_processed_text(args[0], 'aei')
    if len(args) == 3 and args[0] == '-chars':
        print_processed_text(args[2], args[1])
```

get the command line arguments as a list!

```
$ python3 DeleteCharacters.py -chars aei poem.txt
```

Arguments

slice off the first item in the list

```
def main():
    args = sys.argv[1:]
    if len(args) == 1:
        print_processed_text(args[0], 'aei')
    if len(args) == 3 and args[0] == '-chars':
        print_processed_text(args[2], args[1])
```

```
$ python3 DeleteCharacters.py -chars aei poem.txt
```

Arguments

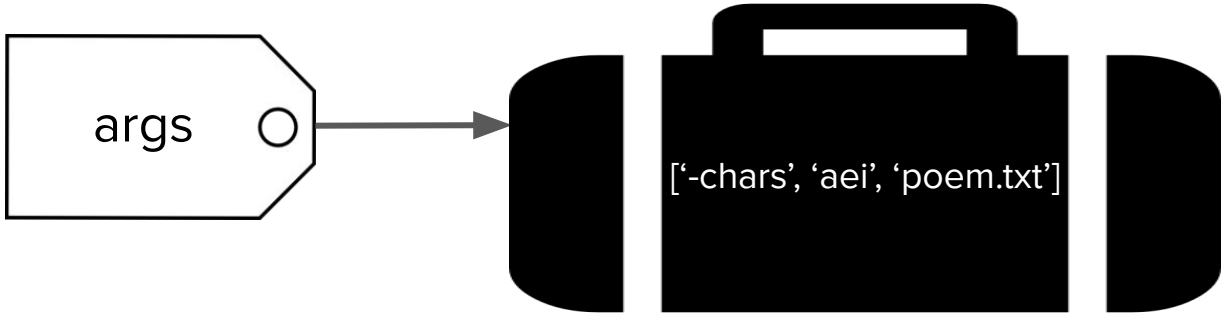
slice off the first item in the list

```
def main():
    args = sys.argv[1:]
    if len(args) == 1:
        print_processed_text(args[0], 'aei')
    if len(args) == 3 and args[0] == '-chars':
        print_processed_text(args[2], args[1])
```

```
$ python3 DeleteCharacters.py -chars aei poem.txt
```

Now our list doesn't include the script name.

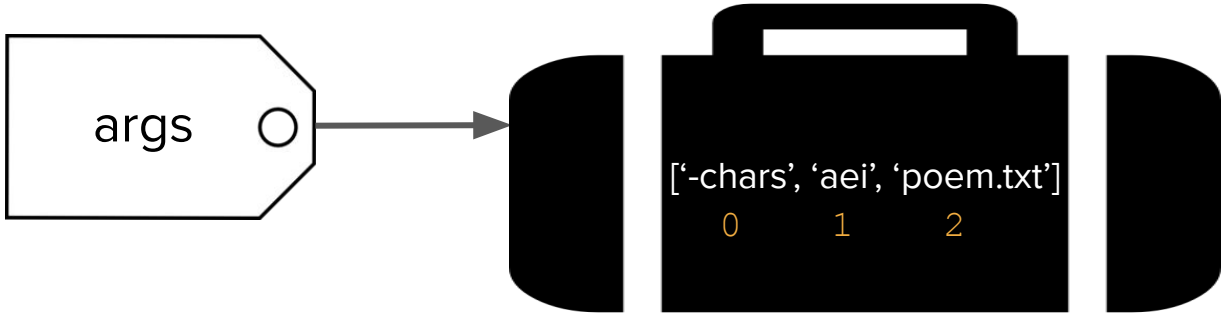
Arguments




```
def main():
    args = sys.argv[1:]
    if len(args) == 1:
        print_processed_text(args[0], 'aei')
    if len(args) == 3 and args[0] == '-chars':
        print_processed_text(args[2], args[1])
```

```
$ python3 DeleteCharacters.py -chars aei poem.txt
```

Arguments




```
def main():
    args = sys.argv[1:]
    if len(args) == 1:
        print_processed_text(args[0], 'aei')
    if len(args) == 3 and args[0] == '-chars':
        print_processed_text(args[2], args[1])
```

```
$ python3 DeleteCharacters.py -chars aei poem.txt
```

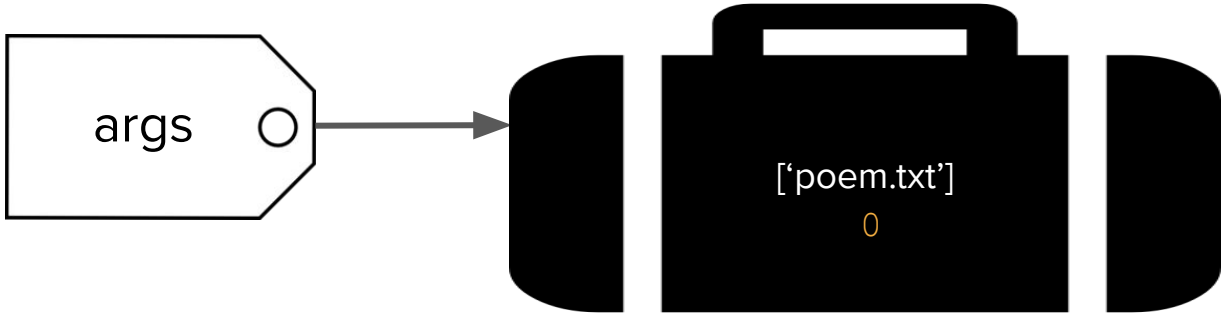
Arguments

```
def main():
    args = sys.argv[1:]
    if len(args) == 1:
        print_processed_text(args[0], 'aei')
    if len(args) == 3 and args[0] == '-chars':
        print_processed_text(args[2], args[1])
```

Think/Pair/Share:
What would args be?
What lines of code would execute?

```
$ python3 DeleteCharacters.py poem.txt
```

Arguments




```
def main():
    args = sys.argv[1:]
    if len(args) == 1:
        print_processed_text(args[0], 'aei')
    if len(args) == 3 and args[0] == '-chars':
        print_processed_text(args[2], args[1])
```

```
$ python3 DeleteCharacters.py poem.txt
```

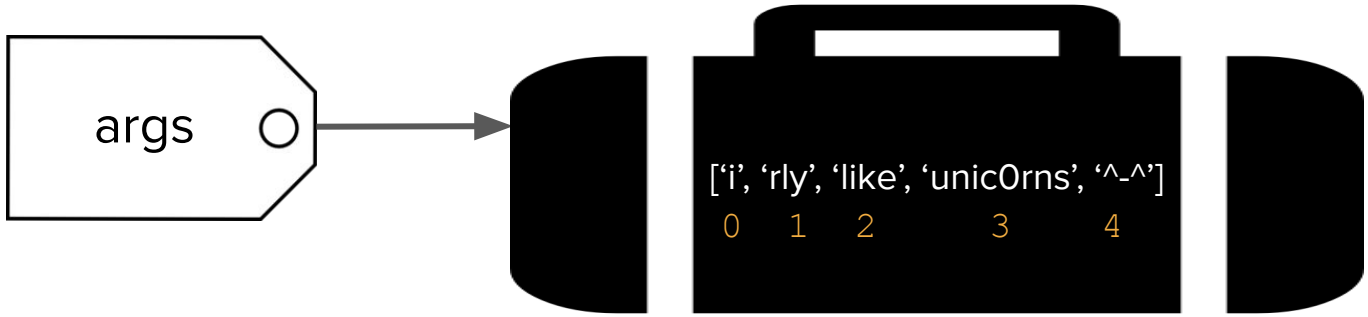
Arguments

```
def main():
    args = sys.argv[1:]
    if len(args) == 1:
        print_processed_text(args[0], 'aei')
    if len(args) == 3 and args[0] == '-chars':
        print_processed_text(args[2], args[1])
```

Think/Pair/Share:
What would args be?

```
$ python3 DeleteCharacters.py i rly like unic0rns ^-^
```

Arguments

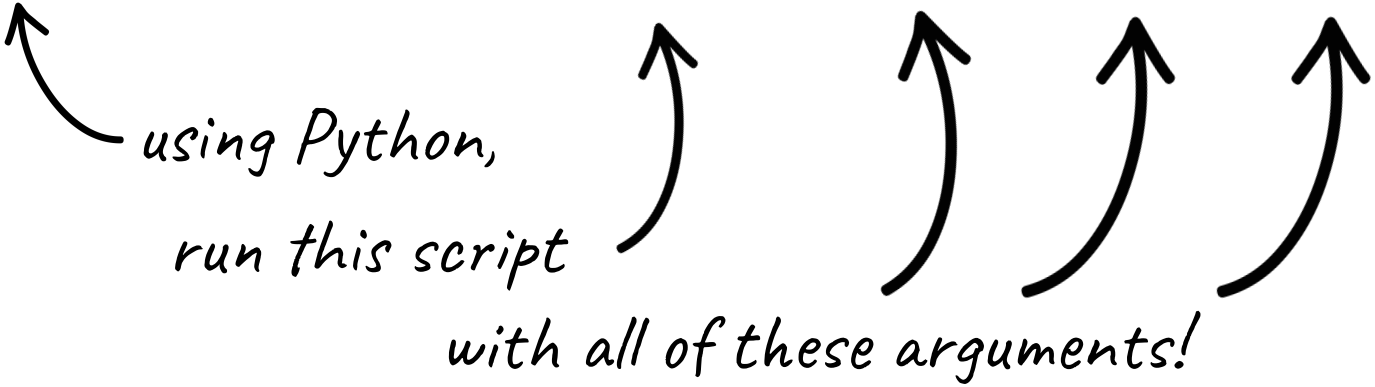



```
def main():
    args = sys.argv[1:]
    if len(args) == 1:
        print_processed_text(args[0], 'aei')
    if len(args) == 3 and args[0] == '-chars':
        print_processed_text(args[2], args[1])
```

```
$ python3 DeleteCharacters.py i rly like unic0rns ^-^
```

Takeaways on arguments

```
python3 DeleteCharacters.py -chars aei poem.txt
```



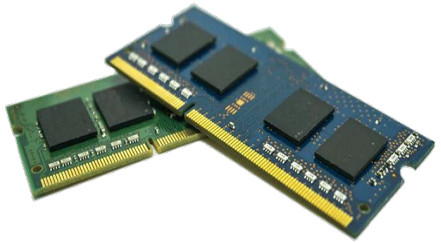
Takeaways on arguments

- We can use sys.argv to get a list of strings that correspond to the command line arguments!

Files

Storing Information

When we're running a program, variables and information are stored on RAM (Random Access Memory)



When we're not running a program and we want to save information, we store it on our hard drive (also called disk)

What’s in a text file?

```
0 The suns are able to fall and rise:
1 When that brief light has fallen for us,
2 we must sleep a never ending night.
```

- No bold/italics!
- Each line is ended by the ‘\n’ newline character!
  - Except for the last line, which doesn’t have a ‘\n’.

What's in a text file?

```
0 The suns are able to fall and rise:\n
1 When that brief light has fallen for us,\n
2 we must sleep a never ending night.
```

- No bold/italics!
- Each line is ended by the '\n' newline character!
  - Except for the last line, which doesn't have a '\n'.

File Reading – catullus.txt

```
0 The suns are able to fall and rise:\n
1 When that brief light has fallen for us,\n
2 we must sleep a never ending night.
```

```
with open('catullus.txt', 'r') as f:
    for line in f:
        print(line)
```

File Reading – catullus.txt

```
0 The suns are able to fall and rise:\n
1 When that brief light has fallen for us,\n
2 we must sleep a never ending night.
```

```
with open('catullus.txt', 'r') as f:
    for line in f:
        print(line)
```

print() automatically adds a \n!

Output:

How can we avoid the extra output line?

```
The suns are able to fall and rise:\n\n
When that brief light has fallen for us,\n\n
we must sleep a never ending night.
```

```
with open('catullus.txt', 'r') as f:
    for line in f:
        print(line)
```

print() automatically adds a '\n'!

Output:

```
The suns are able to fall and rise:\n
When that brief light has fallen for us,\n
we must sleep a never ending night.
```

```
with open('catullus.txt', 'r') as f:
    for line in f:
        print(line, end='')
```

Output:

```
The suns are able to fall and rise:\n
When that brief light has fallen for us,\n
we must sleep a never ending night.
```

```
with open('catullus.txt', 'r') as f:
    for line in f:
        print(line, end='')
```

"once you've printed this line, don't add on a '\n'"

How can I separate valuable data from junk?

Parsing!

```
Geo_FIPS,Geo_GEOID,Geo_NAME,Geo_QName,Geo_STUSAB,Geo_SUMLEV,Geo_GEOCOMP,Geo_FILEID,Geo_LOGRECNO,Geo_US,Geo_REGION,Geo_DIVISION,Geo_STATECE,Geo_STATE,Geo_COUNTY,Geo_COUSUB,Geo_PLACE,Geo_PLACESE,Geo_TRACT,Geo_BLKGRP,Geo_CONCIT,Geo_AIANHH,Geo_AIANHHFP,Geo_AIHHTLI,Geo_AITSCE,Geo_AITS,Geo_ANRC,Geo_CBSA,Geo_CSA,Geo_METDIV,Geo_MACC,Geo_MEMI,Geo_NECTA,Geo_CNECTA,Geo_NECTADIV,Geo_UA,Geo_UACP,Geo_CDCURR,Geo_SLDU,Geo_SLDL,Geo_VTD,Geo_ZCTA3,Geo_ZCTA5,Geo_SUBMCD,Geo_SDELM,Geo_SDSEC,Geo_SDUNI,Geo_UR,Geo_PCI,Geo_TAZ,Geo_UGA,Geo_BTTR,Geo_BTBG,Geo_PUMA5,Geo_PUMA1,SE_A00001_001,SE_A00002_001,SE_A00002_002,SE_A00002_003,SE_A14001_001,SE_A14001_002,SE_A14001_003,SE_A14001_004,SE_A14001_005,SE_A14001_006,SE_A14001_007,SE_A14001_008,SE_A14001_009,SE_A14001_010,SE_A14001_011,SE_A14001_012,SE_A14001_013,SE_A14001_014,SE_A14001_015,SE_A14001_016,SE_A14001_017,SE_A14006_001,SE_A14007_001,SE_A14007_002,SE_A14007_003,SE_A14007_004,SE_A14007_005,SE_A14007_006,SE_A14007_007,SE_A14007_008,SE_A14007_009,SE_A14007_010,SE_A14008_001,SE_A14010_001,SE_A14011_001,SE_A14012_001,SE_A14013_001,SE_A14014_001,SE_A14014_002,SE_A14014_003,SE_A14014_004,SE_A14014_005,SE_A14014_006,SE_A14014_007,SE_A15001_001,SE_A15001_002,SE_A15001_003,SE_A15001_004,SE_A15001_005,SE_A15001_006,SE_A15001_007,SE_A15001_008,SE_A15001_009,SE_A15001_010,SE_A15001_011,SE_A15001_012,SE_A15001_013,SE_A15001_014,SE_A15001_015,SE_A15001_016,SE_A15001_017,SE_A15001_018,SE_A14015_001,SE_A14015_002,SE_A14015_003,SE_A10011_001,SE_A10011_002,SE_A10011_003,SE_A10019_001,SE_A10019_002,SE_A10019_003,SE_A10016_001,SE_A10016_002,SE_A10016_003,SE_A10012_001,SE_A10012_002,SE_A10012_003,SE_A10017_001,SE_A10017_002,SE_A10017_003,SE_A10018_001,SE_A10018_002,SE_A10018_003,SE_A10014_001,SE_A10014_002,SE_A10014_003,SE_A10015_001,SE_A10015_002,SE_A10015_003,SE_A10013_001,SE_A10013_002,SE_A10013_003,SE_A14024_001,SE_A14024A_001,SE_A14024B_001,SE_A14024C_001,SE_A14024D_001,SE_A14024E_001,SE_A14024F_001,SE_A14024G_001,SE_A14024H_001,SE_A14024I_001
"06085511500","14000US06085511500","Census Tract 5115, Santa Clara County, California","Census Tract 5115, Santa Clara County, California","ca","140","00","ACSSF","0009922",,,,,"06","085",,,,"511500",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,8104,8104,5770.712,1.40433276138731,3051,93,63,88,49,20,66,57,97,6,70,188,343,236,135,412,1128,154917,154917,142000,,97321,168906,,,175938,,153917,200032.841691249,205393,248774.015345269,84231,112728.467153285,61921,106620,165365,35179,43281,95106,16544,91838,,,22389,70096,119145,134203,,60571,23371,112688,147188,70288,,5125,21750,51797,89510,154917,211985,89620,3051,2529,522,3051,2393,658,3051,722,2329,3051,1560,1491,3051,684,2367,3051,73,2978,3051,23,3028,3051,461,2590,3051,365,2686,77024,85502,20642,51995,65458,,94733,49420,67003,87497
"06085511608","14000US06085511608","Census Tract 5116.08, Santa Clara County, California","Census Tract 5116.08, Santa Clara County, California","ca","140","00","ACSSF","0009923",,,,,"06","085",,,,"511608",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,3181,3181,3355.507,0.947993967539618,102,7,16,0,0,4,15,10,11,0,0,10,19,4,6,0,0,37250,37250,32250,,,61500,,,,,32250,49602.9411764706,,,38750,52600,4340,4852,47794,4281,3808,33523,3490,35658,,,,34375,42500,42750,,,,37143,43000,31208,,,,31875,,37250,,36750,102,95,7,102,95,7,102,0,102,102,20,82,102,16,86,102,0,102,102,0,102,102,0,102,102,29,73,8761,8834,4353,3850,11216,,4249,6956,9156,8053
"06085511609","14000US06085511609","Census Tract 5116.09, Santa Clara County, California","Census Tract 5116.09, Santa Clara County, California","ca","140","00","ACSSF","0009924",,,,,"06","085",,,,"511609",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,3363,3363,5646.582,0.595581523929841,1727,105,44,74,10,18,20,29,12,8,89,235,231,132,126,174,420,98083,98083,106118,65977,,98083,,,2499,,113375,176693.341053851,132000,237644.846796657,79750,133187.611496531,77500,87938,119583,63250,66659,88750,35625,82458,,22000,66307,63611,84529,83906,,,106875,125000,83242,71683,,,65625,57000,85739,98083,,100682,1727,1169,558,1727,1162,565,1727,81,1646,1727,713,1014,1727,490,1237,1727,37,1690,1727,0,1727,1727,192,1535,1727,83,1644,93029,112971,45865,,53141,,23976,27185,27193,120451
"06085513000","14000US06085513000","Census Tract 5130, Santa Clara County, California","Census Tract 5130, Santa Clara County, California","ca","140","00","ACSSF","0009992",,,,,"06","085",,,,"513000",,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,8691,8691,8176.084,1.06297828406927,2313,365,108,95,45,61,372,217,92,59,128,165,233,108,56,96,113,37361,37361,38943,76066,,35532,,,29632,47692,38155,62777.9939472546,113500,132696.354166667,34637,47823.3799896319,16677,20703,37539,11344,13906,39311,9066,35900,,,,34592,36696,36470,,,,39583,35504,33953,,,,31250,39133,37361,250001,36912,2313,1963,350,2313,1963,350,2313,125,2188,2313,357,1956,2313,15,2298,2313,60,2253,2313,15,2298,2313,0,2313,2313,294,2019,21796,22192,19828,13653,24451,,17889,9980,14821,23362
```


Data from Social Explorer: ACS 2017

Statistics	Census Tract 5115, Santa Clara County, California		Census Tract 5116.08, Santa Clara County, California		Census Tract 5116.09, Santa Clara County, California		Census Tract 5130, Santa Clara County, California		TOTAL (All Selected Census Tracts)	
SE:A00001. Total Population										
Total Population	8,104		3,181		3,363		8,691		23,339	
SE:A00002. Population Density (Per Sq. Mile)										
Total Population	8,104		3,181		3,363		8,691		23,339	
Population Density (Per Sq. Mile)	5,770.7		3,355.5		5,646.6		8,176.1		5,818.9	
Area (Land)	1.40		0.95		0.60		1.06		4.01	
SE:A14001. Household Income (In 2017 Inflation Adjusted Dollars)										
Households:	3,051		102		1,727		2,313		7,193	
Less than $10,000	93	3.1%	7	6.9%	105	6.1%	365	15.8%	570	7.9%
$10,000 to $14,999	63	2.1%	16	15.7%	44	2.6%	108	4.7%	231	3.2%
$15,000 to $19,999	88	2.9%	0	0%	74	4.3%	95	4.1%	257	3.6%
$20,000 to $24,999	49	1.6%	0	0%	10	0.6%	45	2.0%	104	1.5%
$25,000 to $29,999	20	0.7%	4	3.9%	18	1.0%	61	2.6%	103	1.4%
$30,000 to $34,999	66	2.2%	15	14.7%	20	1.2%	372	16.1%	473	6.6%
$35,000 to $39,999	57	1.9%	10	9.8%	29	1.7%	217	9.4%	313	4.4%
$40,000 to $44,999	97	3.2%	11	10.8%	12	0.7%	92	4.0%	212	3.0%
$45,000 to $49,999	6	0.2%	0	0%	8	0.5%	59	2.6%	73	1.0%
$50,000 to $59,999	70	2.3%	0	0%	89	5.2%	128	5.5%	287	4.0%
$60,000 to $74,999	188	6.2%	10	9.8%	235	13.6%	165	7.1%	598	8.3%
$75,000 to $99,999	343	11.2%	19	18.6%	231	13.4%	233	10.1%	826	11.5%
$100,000 to $124,999	236	7.7%	4	3.9%	132	7.6%	108	4.7%	480	6.7%
$125,000 to $149,999	135	4.4%	6	5.9%	126	7.3%	56	2.4%	323	4.5%
$150,000 to $199,999	412	13.5%	0	0%	174	10.1%	96	4.2%	682	9.5%
$200,000 or More	1,128	37.0%	0	0%	420	24.3%	113	4.9%	1,661	23.1%

What is data?

```
$GPGGA,005328.000,3726.1389,N,12210.2515,W,2,07,1.3,22.5,M,-25.7,M,2.0,0000*70

$GPGSA,M,3,09,23,07,16,30,03,27,,,,,2.3,1.3,1.9*38

$GPRMC,005328.000,A,3726.1389,N,12210.2515,W,0.00,256.18,221217,,,D*78

$GPGGA,005329.000,3726.1389,N,12210.2515,W,2,07,1.3,22.5,M,-25.7,M,2.0,0000*71

$GPGSA,M,3,09,23,07,16,30,03,27,,,,,2.3,1.3,1.9*38

$GPRMC,005329.000,A,3726.1389,N,12210.2515,W,0.00,256.18,221217,,,D*79

$GPGGA,005330.000,3726.1389,N,12210.2515,W,2,07,1.3,22.5,M,-25.7,M,3.0,0000*78

$GPGSA,M,3,09,23,07,16,30,03,27,,,,,2.3,1.3,1.9*38
```

Read more about NMEA

What is data?

- Usually just text!
    - Text is a common data exchange format.

Parsing

Definition

Parsing
The act of reading “raw” text and converting it into a more useful format stored in memory.

Components of Parsing

Components of Parsing

- File Reading

Components of Parsing

- File Reading
- String Manipulation

Components of Parsing

- File Reading
- String Manipulation
- Advanced Control Flow

Components of Parsing

- File Reading
- String Manipulation
- Advanced Control Flow
- Container Data Types

Components of Parsing

- File Reading
- **String Manipulation**
- **Advanced Control Flow**
- Container Data Types

String Manipulation - Useful Functions

```
s.isalpha()

s.isdigit()

s.isspace()
```

String Manipulation - Useful Functions

```
s.isalpha()

s.isdigit()

s.isspace()
```

String Manipulation - Useful Functions

```
s.isalpha()

s.isdigit()

s.isspace()
```

applies to spaces, tabs, and newlines.
Tabs are written '\t'. Newlines are '\n'.

String Manipulation - Useful Functions

String Manipulation - Useful Functions

```
s.startswith(substr)

s.endswith(substr)
```

String Manipulation - Useful Functions

```
s.startswith(substr)

s.endswith(substr)
```


```
>>> 'Sonja'.startswith('Son')
```

String Manipulation - Useful Functions

```
s.startswith(substr)

s.endswith(substr)
```


```
>>> 'Sonja'.startswith('Son')

True
```

String Manipulation - Useful Functions

```
>>> s = 'computer'
```

String Manipulation - Useful Functions

```
>>> s = 'computer'

>>> 'put' in s
```

String Manipulation - Useful Functions

```
>>> s = 'computer'

>>> 'put' in s
```

You can use **in** with strings, like lists!

String Manipulation - Useful Functions

```
>>> s = 'computer'

>>> 'put' in s

True
```

You can use **in** with strings, like lists!

String Manipulation - Useful Functions

```
>>> s = 'computer'

>>> 'put' in s

True
```

String Manipulation - Useful Functions

```
>>> s = 'hello!'
```

String Manipulation - Useful Functions

```
>>> s = 'hello!'

>>> s.find('!')
```

String Manipulation - Useful Functions

```
>>> s = 'hello!'

>>> s.find('!')

5
```

find() returns the index of the first occurrence of the substring you pass in

String Manipulation - Useful Functions

```
>>> s = 'hello!'

>>> s.find('!')

5

>>> s.find('l')
```

find() returns the index of the first occurrence of the substring you pass in

String Manipulation - Useful Functions

```
>>> s = 'hello!'

>>> s.find('!')

5

>>> s.find('l')

2
```

find() returns the index of the **first** occurrence of the substring you pass in

String Manipulation - Useful Functions

```
>>> s = 'hello!'

>>> s.find('w')
```

String Manipulation - Useful Functions

```
>>> s = 'hello!'

>>> s.find('w')

-1
```

if the string doesn't contain the substring, return -1

String Manipulation - Useful Functions

```
>>> s = 'hello!'

>>> s.find('w')

-1

>>> s.find('l', 3)
```

optionally can pass in start index (or end index)

String Manipulation - Useful Functions

```
>>> s = 'hello!'

>>> s.find('w')

-1

>>> s.find('l', 3)

3
```

optionally can pass in start index (or end index)

String Manipulation - Useful Functions

```
>>> s = 'hello!'

>>> s.find('w')

-1

>>> s.find('l', 3)

3
```

the format is:

s.find(substr, start_index, end_index)

String Manipulation - Useful Functions

```
>>> s = 'hello!'

>>> s.find('w')

-1

>>> s.find('l', 3)

3
```

the format is:

s.find(substr, start_index, end_index)

Think/Pair/Share:

Find the first ‘@’ in s. Return the substring made of 0 or more alpha characters following the ‘@’.

String Manipulation - Useful Functions

```
>>> s = ‘ hello world! ’
```

String Manipulation - Useful Functions

```
>>> s = ' hello world! '
>>> s.strip()
```

removes whitespace on left & right sides of string

String Manipulation - Useful Functions

```
>>> s = ' hello world! '
>>> s.strip()
```

removes whitespace on left & right sides of string

String Manipulation - Useful Functions

```
>>> s = ' hello world! '
>>> s.strip()
'hello world!'
```

removes whitespace on left & right sides of string

String Manipulation - Useful Functions

```
>>> s = ' hello world! '
>>> s.strip()
'hello world!'
>>> s = ' hello world!\n '
```

removes whitespace on left & right sides of string

String Manipulation - Useful Functions

```
>>> s = ' hello world! '
>>> s.strip()
'hello world!'
>>> s = ' hello world!\n '
>>> s.strip()
```

removes whitespace on left & right sides of string

can be used on newlines and tabs as well as spaces

String Manipulation - Useful Functions

```
>>> s = ' hello world! '
>>> s.strip()
'hello world!'
>>> s = ' hello world!\n '
>>> s.strip()
'hello world!'
```

removes whitespace on left & right sides of string

can be used on newlines and tabs as well as spaces

String Manipulation - Useful Functions

```
>>> s = ' hello world! '
>>> s.strip()
'hello world!'
>>> s = ' hello world!\n '
>>> s.strip()
'hello world!'
```

removes whitespace on left & right sides of string

can be used on newlines and tabs as well as spaces

Recall: (output)

How can we avoid the extra output line?

```
The suns are able to fall and rise:\n\n
When that brief light has fallen for us,\n\n
we must sleep a never ending night.
```

```
with open('catullus.txt', 'r') as f:
    for line in f:
        print(line)
```

print() automatically adds a '\n'!

Recall: (output)

How can we avoid the extra output line?

```
The suns are able to fall and rise:\n
When that brief light has fallen for us,\n
we must sleep a never ending night.
```

```
with open('catullus.txt', 'r') as f:
    for line in f:
        line = line.strip()
        print(line)
```

How do we represent strings?

- Google “omega uppercase unicode”
  - ‘03A9’

How do we represent strings?

- Google “omega uppercase unicode”
  - ‘03A9’
  - hexadecimal notation (base-16) = 0-9 plus letters A-F

How do we represent strings?

- Google “omega uppercase unicode”
  - ‘03A9’
  - hexadecimal notation (base-16) = 0-9 plus letters A-F

```
>>> s = '\u03A9'
```

How do we represent strings?

- Google “omega uppercase unicode”
  - ‘03A9’
  - hexadecimal notation (base-16) = 0-9 plus letters A-F

```
>>> s = '\u03A9'

>>> s
```

How do we represent strings?

- Google “omega uppercase unicode”
  - ‘03A9’
  - hexadecimal notation (base-16) = 0-9 plus letters A-F

```
>>> s = '\u03A9'

>>> s

'Ω'
```

Components of Parsing

- File Reading
- **String Manipulation**
- **Advanced Control Flow**
- Container Data Types

Compound Boolean Expressions

```
s = 'yay'

if len(s) == 2 and s[1] == 'a':

    # do something
```

Compound Boolean Expressions

Compound Boolean Expressions

```
s = 'yay'
if len(s) == 2 and s[1] == 'a':
    # do something
```

Compound Boolean Expressions

```
s = 'yay'
if len(s) == 2 and s[1] == 'a':
    # do something
```

This will never get executed!

This is also known as "shortcircuiting".

Why is this useful?

Why is this useful?

```
s = ‘’

if len(s) != 0 and s[0] == ‘a’:

    # do something
```

Why is this useful?

```
s = ''
if len(s) != 0 and s[0] == 'a':
    # do something
```

Advanced Control Flow

break

continue

Advanced Control Flow

```
# print words in all_words until hit a censored word!
```

Advanced Control Flow

```
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            break
        print(word)
```

Advanced Control Flow

```
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            break
        print(word)
```

Advanced Control Flow

```
# print words in all_words that aren't censored!
```

Advanced Control Flow

```
# print words in all_words that aren't censored!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            continue
        print(word)
```

Advanced Control Flow

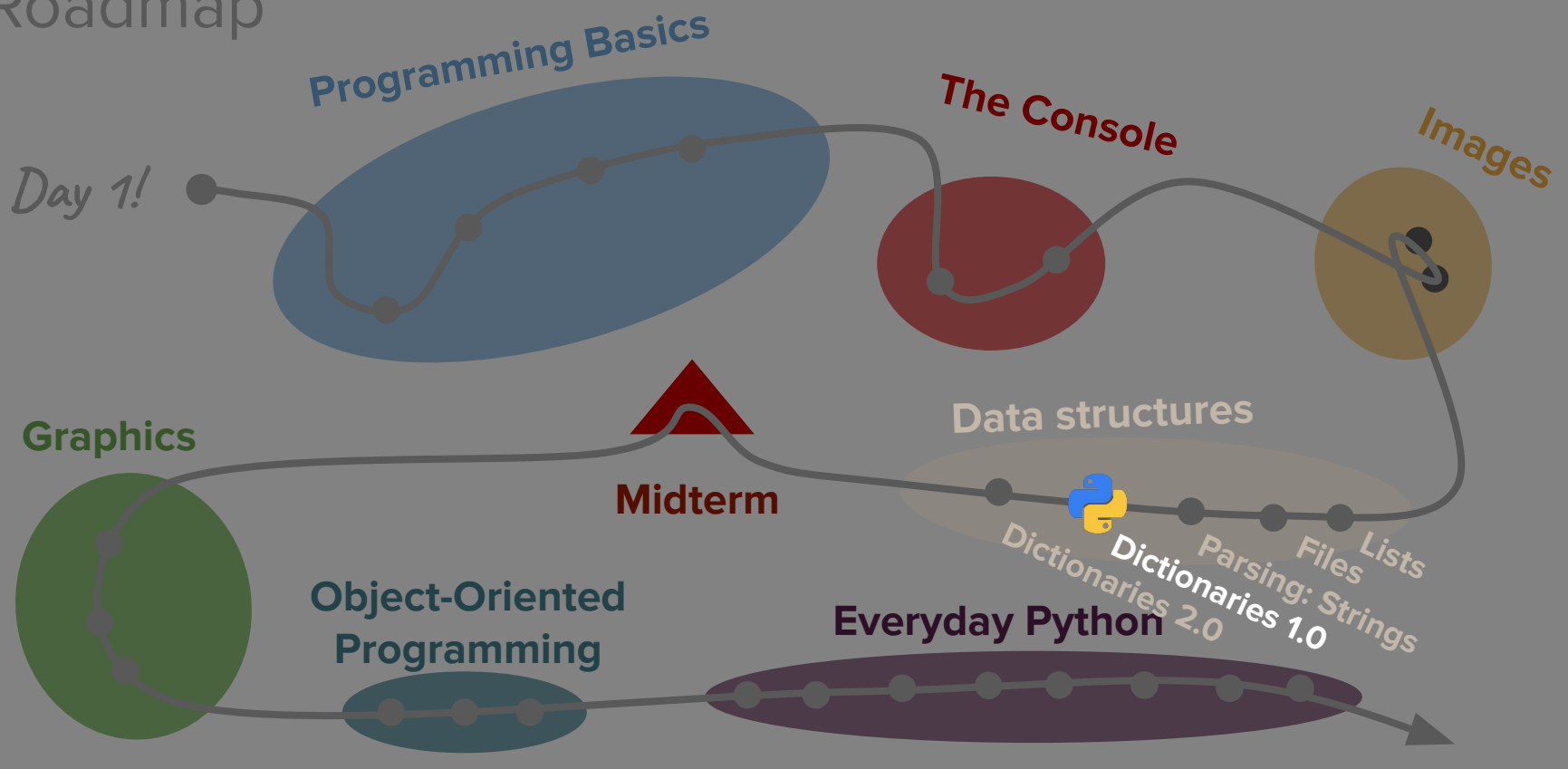
```
# print words in all_words that aren't censored!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            continue
        print(word)
```

Think/Pair/Share:

Print list of zoo animals (not including the bears) and corresponding list of number of times each animal has been fed.

What’s next?

Roadmap


Programming Basics
Day 1!
The Console
Images
Graphics
Midterm
Data structures
Lists
Files
Parsing: Strings
Dictionaries 1.0
Dictionaries 2.0
Object-Oriented Programming
Everyday Python
Life after CS106AP!

What’s next?

- **Dictionaries**
  - Is there a better way to store complex data?