BASIC SKILLS FOR A COMPUTER PROGRAMMER

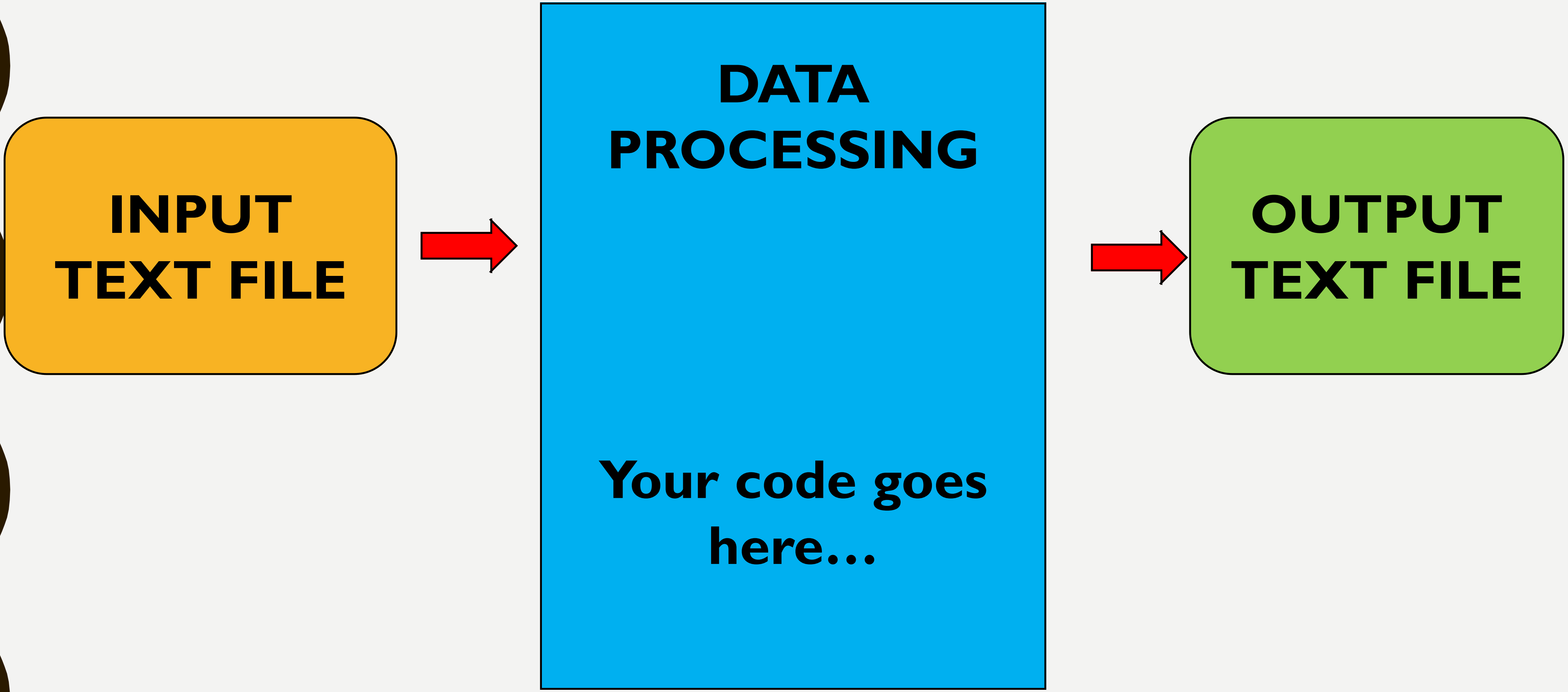
AN INTRODUCTION TO PYTHON SYNTAX

KATHY BREEN 2017

THE BARE MINIMUM SKILL SET

- Reading/writing files
- Data type literacy and conversion
- Arithmetic and logical operators
- String manipulation
- For loops
- If/else statements

WORKFLOW



INPUT TEXT FILE

DATA PROCESSING

Your code goes here…

OUTPUT TEXT FILE

TEXT FILES AND TEXT EDITORS

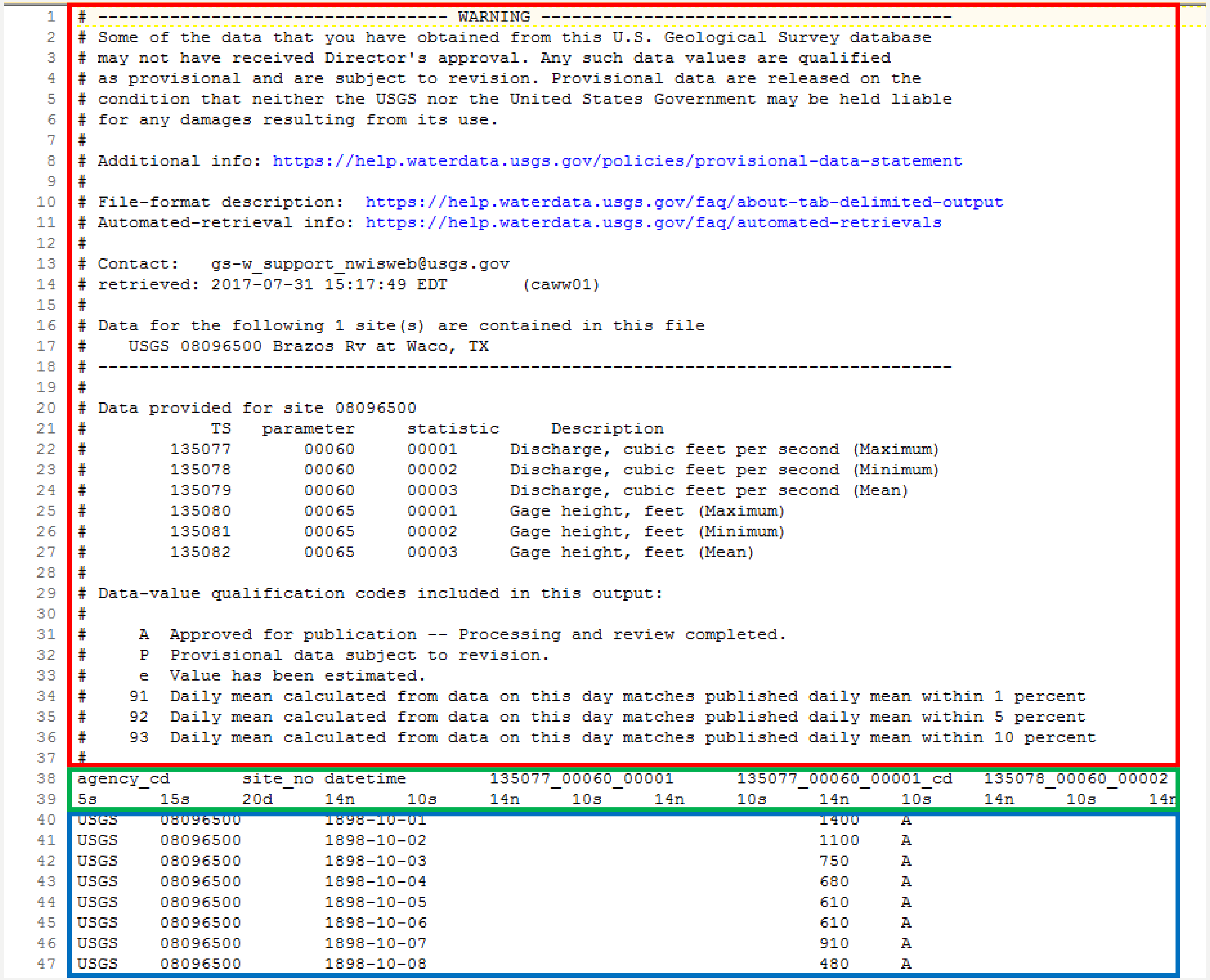
EMACS

SUBLIME

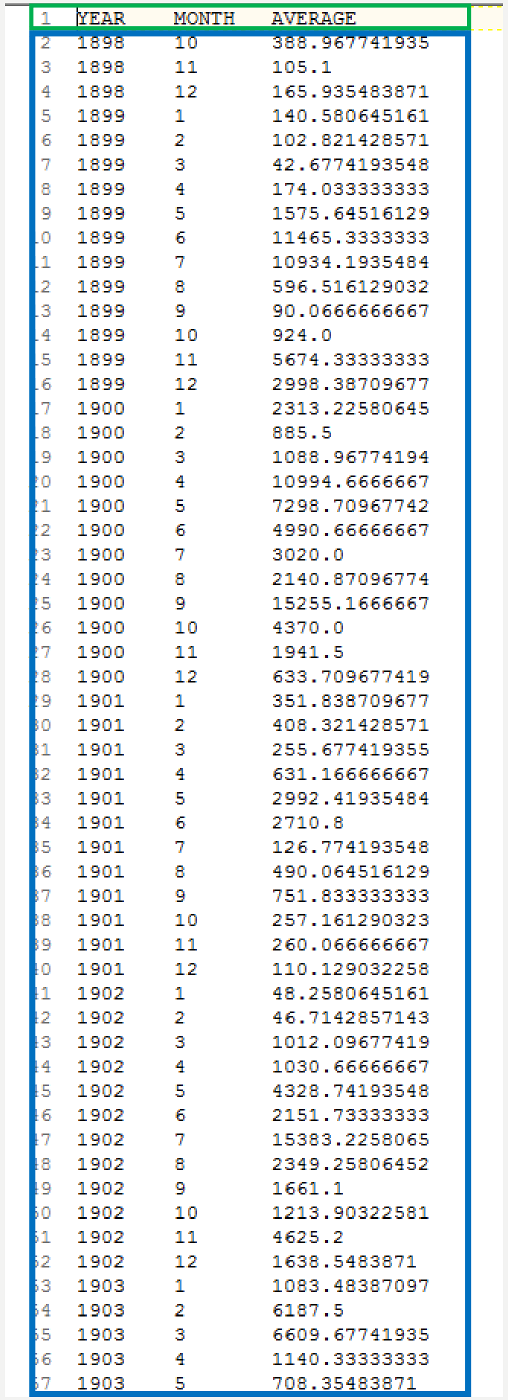
🚫 MICROSOFT WORD

NOTEPAD++

TEXTPAD



```
# ---------------------------------- WARNING ----------------------------------------
# Some of the data that you have obtained from this U.S. Geological Survey database
# may not have received Director's approval. Any such data values are qualified
# as provisional and are subject to revision. Provisional data are released on the
# condition that neither the USGS nor the United States Government may be held liable
# for any damages resulting from its use.
#
# Additional info: https://help.waterdata.usgs.gov/policies/provisional-data-statement
#
# File-format description:  https://help.waterdata.usgs.gov/faq/about-tab-delimited-output
# Automated-retrieval info: https://help.waterdata.usgs.gov/faq/automated-retrievals
#
# Contact:   gs-w_support_nwisweb@usgs.gov
# retrieved: 2017-07-31 15:17:49 EDT       (caww01)
#
# Data for the following 1 site(s) are contained in this file
#    USGS 08096500 Brazos Rv at Waco, TX
# -----------------------------------------------------------------------------------
#
# Data provided for site 08096500
#            TS   parameter     statistic     Description
#        135077       00060     00001     Discharge, cubic feet per second (Maximum)
#        135078       00060     00002     Discharge, cubic feet per second (Minimum)
#        135079       00060     00003     Discharge, cubic feet per second (Mean)
#        135080       00065     00001     Gage height, feet (Maximum)
#        135081       00065     00002     Gage height, feet (Minimum)
#        135082       00065     00003     Gage height, feet (Mean)
#
# Data-value qualification codes included in this output:
#
#     A  Approved for publication -- Processing and review completed.
#     P  Provisional data subject to revision.
#     e  Value has been estimated.
#    91  Daily mean calculated from data on this day matches published daily mean within 1 percent
#    92  Daily mean calculated from data on this day matches published daily mean within 5 percent
#    93  Daily mean calculated from data on this day matches published daily mean within 10 percent
#
agency_cd      site_no datetime        135077_00060_00001      135077_00060_00001_cd      135078_00060_00002
5s      15s      20d      14n      10s      14n      10s      14n      10s      14n      10s      14n      10s      14r
USGS    08096500        1898-10-01                                      1400    A
USGS    08096500        1898-10-02                                      1100    A
USGS    08096500        1898-10-03                                      750     A
USGS    08096500        1898-10-04                                      680     A
USGS    08096500        1898-10-05                                      610     A
USGS    08096500        1898-10-06                                      610     A
USGS    08096500        1898-10-07                                      910     A
USGS    08096500        1898-10-08                                      480     A
```

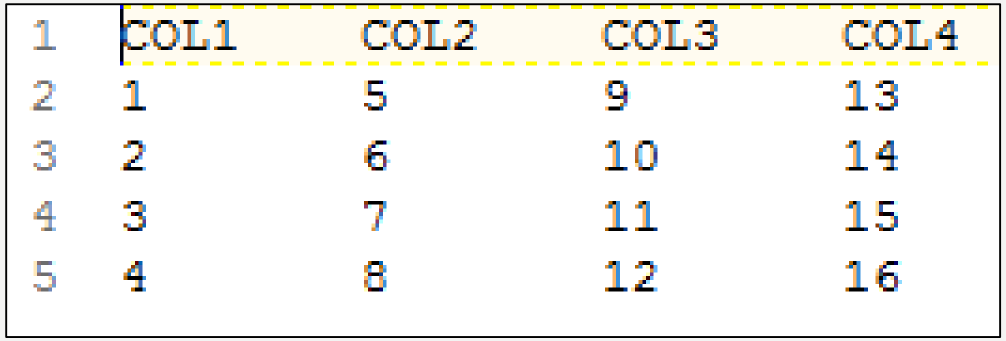


```
YEAR    MONTH   AVERAGE
1898    10      388.967741935
1898    11      105.1
1898    12      165.935483871
1899    1       140.580645161
1899    2       102.821428571
1899    3       42.6774193548
1899    4       174.033333333
1899    5       1575.64516129
1899    6       11465.3333333
1899    7       10934.1935484
1899    8       596.516129032
1899    9       90.0666666667
1899    10      924.0
1899    11      5674.33333333
1899    12      2998.38709677
1900    1       2313.22580645
1900    2       885.5
1900    3       1088.96774194
1900    4       10994.6666667
1900    5       7298.70967742
1900    6       4990.66666667
1900    7       3020.0
1900    8       2140.87096774
1900    9       15255.1666667
1900    10      4370.0
1900    11      1941.5
1900    12      633.709677419
1901    1       351.838709677
1901    2       408.321428571
1901    3       255.677419355
1901    4       631.166666667
1901    5       2992.41935484
1901    6       2710.8
1901    7       126.774193548
1901    8       490.064516129
1901    9       751.833333333
1901    10      257.161290323
1901    11      260.066666667
1901    12      110.129032258
1902    1       48.2580645161
1902    2       46.7142857143
1902    3       1012.09677419
1902    4       1030.66666667
1902    5       4328.74193548
1902    6       2151.73333333
1902    7       15383.2258065
1902    8       2349.25806452
1902    9       1661.1
1902    10      1213.90322581
1902    11      4625.2
1902    12      1638.5483871
1903    1       1083.48387097
1903    2       6187.5
1903    3       6609.67741935
1903    4       1140.33333333
1903    5       708.35483871
```

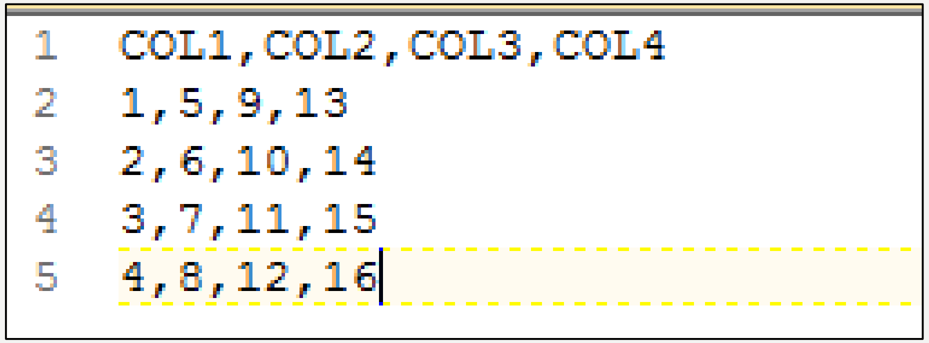
DELIMITERS

Tab-delimited
test.txt

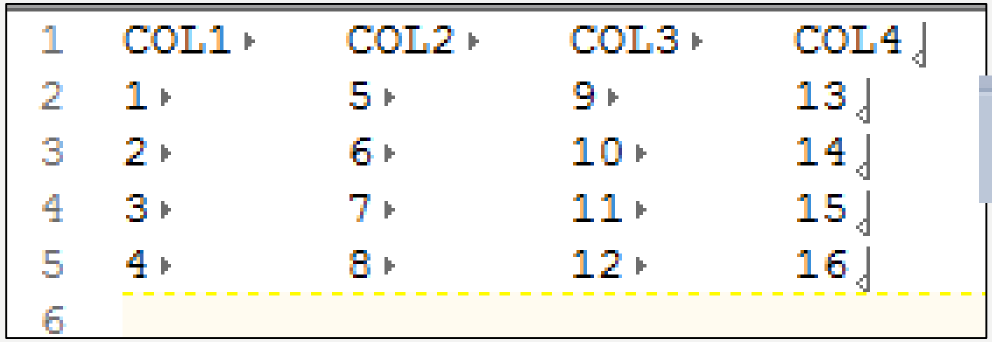
Comma-delimited
text.csv



```
COL1      COL2      COL3      COL4
1         5         9         13
2         6         10        14
3         7         11        15
4         8         12        16
```

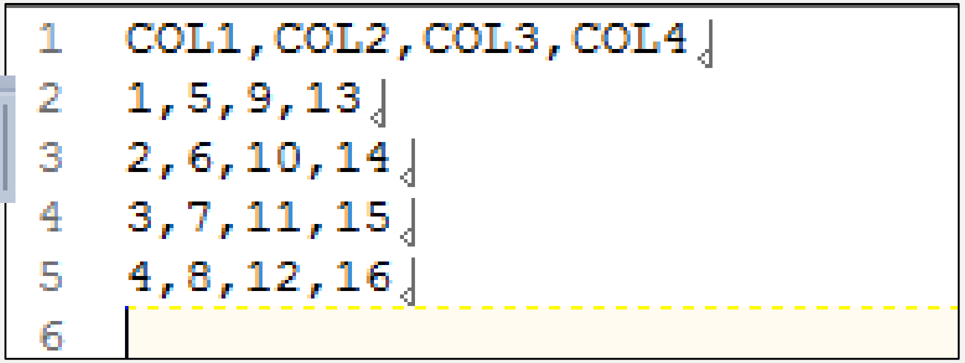


```
COL1,COL2,COL3,COL4
1,5,9,13
2,6,10,14
3,7,11,15
4,8,12,16
```



```
COL1      COL2      COL3      COL4
1         5         9         13
2         6         10        14
3         7         11        15
4         8         12        16

```



```
COL1,COL2,COL3,COL4
1,5,9,13
2,6,10,14
3,7,11,15
4,8,12,16

```

DATA TYPES: CHARACTERS AND STRINGS

A **character** is any single number, letter, or symbol (0-9,a-z,A-Z,!,@,#,etc…) encoded to appear on your screen as you would type it with your keyboard. In most programming languages a character is defined by surrounding it with single or double quotes.

EX: `‘1’` is the character representation of the number one

A **string** is a group of characters, including spaces and punctuation marks.

EX: `‘Hello World’` is a string of 11 characters (H-e-l-l-o-space-W-o-r-l-d)

NOTE: Characters and strings are case-sensitive, which means that capital letters are not equal to their lower-case equivalents.

EX: `‘HELLO WORLD’` is not equal to `‘Hello World’`

DATA TYPES: INTEGERS AND FLOATS

An **integer** is a whole number with no decimal point.

A **floating point number** (float for short) is any number with a decimal point.

EX: The number `1` is an **integer**

The number `1.0` is a **float**

Float to Integer Conversion	Integer to Float Conversion
`int(3.1415927)  = 3`	`float(1) = 1.0`
• Rounding - Can you imagine using pi without the infinite decimal? • Loss of precision	• Increased precision • Requires more memory for storage, but worth it

DATA TYPES: BOOLEANS

A **boolean** is a special type of variable that can have one of two values: `True` or `False`. Booleans are commonly used to make decisions in a logical statement, or to test equality between two statements.

EX:

```
    1 == 1
>>> True
  1 == ‘1’
>>> False
```

`True and True is True`
`True and False is False`
`False and False is False`
`True or True is True`
`True or False is True`
`False or False is False`

DATA TYPES: LIST

A **list** is a one dimensional series of numbers and or strings. In Python, data types can be mixed in a list, which isn't true for all programming languages. A list can be defined in two ways:

```
lst = [1,2,3,'four']
lst = list(1,2,3,'four')
```

Individual list items can be accessed using the slice operator [ ].

```
    elem2 = lst[1]
>>> 2
```

DATA TYPES: DICTIONARY

A **dictionary** is a Python structure for data storage defined by key:value pairs. A **key** is normally a string that describes the value in some way. A **value** can be any data type. To add values to a dictionary or a list, it must be defined first. A dictionary can be defined in two ways:

```
        dct = dict()

        dct = {}
```

Let's say we want to store information for student final grades in a dictionary. We will store the student's first name, last name, and grade.

```
firstname = 'John'
lastname = 'Smith'
finalgrade = 'B+'
student_grades = {'first_name': firstname, 'last_name': lastname,
'grade': finalgrade}
>>> {'first_name': 'John',
     'last_name': 'Smith',
     'grade': 'B+'}
```

DON’T MIX AND MATCH - CONVERT

	EXAMPLE 1	EXAMPLE 2

INPUTS	`str_var = ‘Output: ’`
	`num_var = 2`
		`denominator = ‘100’`
		`numerator = 90.0`

INPUT TYPE	`string` `integer`	`string` `float`
DESIRED OUTPUT	`‘Output: 2’`	`0.9`
OUTPUT TYPE	`string`	`float`
CONVERSION	`integer to string`	`string to float`
CODE	`str_var + str(num_var)`	`numerator / float(denominator)`

ARITHMETIC AND LOGICAL OPERATORS

Arithmetic Operators		
Symbol	Operation	Example
+	Addition	1 + 2 = 3
-	Subtraction	3 - 2 = 1
*	Multiplication	2 * 3 = 6
/	Division	6 / 2 = 3
**	Exponents	4 ** 2 = 16
%	Modulo*	12 % 5 = 2

Logical Operators		
Keyword	Symbol	Statement
and	&	__ and __ are True
or	|	__ or __ are True
is	==	__ is equal to __
not	!=	__ is not equal to __

*Think of a modulo as a remainder. In the example, 2 is the remainder when 12 is divided by 5.

FOR LOOPS

For loops are a control flow statement that allow the user to repeat the same lines of code on multiple elements within the same data structure. This process is called **iteration**. The following code example will square each number in the list **`input`** and store the value in the list **`output`**.

```
input = [1,2,3,4]
output = []
for elem in input:
     squared = elem**2
     output.append(elem)
print output
  >>> [1,4,9,16]
```

control flow statement

iteration code block

no more indent = exit loop

IF…ELIF…ELSE STATEMENTS

if ‘1’ is str(1):
    print ‘if string’
	if ‘1’ is not str(1):
	    print ‘not if string’
	elif ‘1’ is str(1):
	    print ‘elif string’
		if ‘1’ is not str(1):
		    print ‘not if string’
		elif ‘2’ is str(1):
		    print ‘not elif string’
		else:
		    print ‘or else’

if statement evaluates to True Output: ‘if string’	if statement evaluates to False elif statement evaluates to True Output: ‘elif string’	if statement evaluates to False elif statement evaluates to False else is the only option Output: ‘or else’

```
# this code will iterate over the array named input and separate the
values into lists of even or odd numbers

# define input and output lists
input = [1,2,3,4,5,6,7,8,9,10]
evens = []
odds = []

# iterate over inputs
for elem in input:
    # if the element is even ...
    if elem%2 is 0:
        evens.append(elem)
    # if the number isn't even, then it's odd ...
    else:
        odds.append(elem)


>>> [2,4,6,8,10] ← evens

>>> [1,3,5,7,9] ← odds
```

IMPLEMENTATION ALGORITHMS

Implementation algorithm for calculating an average of a list of numbers:

Algorithm	Code
Define input list	`input = [1,2,3,4,5]`
Define the length of the list (number of values)	`N = len(input)`
Calculate sum of input values	`sum_input = sum(input)`
Divide the sum by the number of values	`avg_input = sum_input/N`

```
# define input list
input = [1,2,3,4,5]
# define length of list
N = len(input)
# calculate sum of input values
sum_input = sum(input)
# divide the sum by the number of values
avg_input = sum_input/N
>>> 3
```

RESOURCES

- **Codecademy** https://www.codecademy.com/

Free, interactive, comprehensive way to learn the basics for some of the more popular open source programming languages (Python, Java, command line, SQL…)

- **Documentation** https://www.python.org/

Run through tutorials, read the "Get Started" guide, and browse through libraries of functions you may commonly use.

- **User forums** https://stackexchange.com/

Almost every question you could think to ask has been answered – all you have to do is read through the forums.

Google Search Format: [program name] [version] [problem in ten words or less]

EX: Python 3.6 beginner tutorials

The above Google search will return pages that are ideal for a beginner interested in learning Python…

CONTACT INFO

Kathy Breen

Kathy_Breen@baylor.edu

Office: C460R (main lab)

Need help? Make an appointment on my Outlook calendar – it's always updated. I'm happy to help you get started and grease the wheels when you get stuck.

I use the following languages, some (much) better than others:

- MatLab
- Python
- Javascript (HTML, CSS, jQuery)
- R
- Perl
- SAS
- Fortran