

# CIS 2348 UNIVERSITY OF HOUSTON INFORMATION SYSTEM APPLICATION DEVELOPMENT

FALL 2021

# CHAPTER 8

## STRINGS

# STRINGS REVISITED

- SETS STAGE FOR MORE COMPLEX OBJECTS
- ALLOWS TO SEE THE OPERATIONS INTUITIVELY

# STRING SLICING

SYNTAX:

MY\_SUBSTRING=MYSTRING[A:B]

A IS THE STARTING CHARACTER, B IS FIRST NOT INCLUDED

## SLICING EXAMPLES

```
MY_STRING='THIS IS MY STRING'
```

```
SUBSTRING=MY_STRING[5:7]
```

```
SUBSTRING1=MY_STRING[:5]
```

```
SUBSTRING2=MY_STRING[-6:]
```

```
PRINT(SUBSTRING, SUBSTRING1, SUBSTRING2)
```

# FIND AND REPLACE

- FIND AND REPLACE METHODS OPERATE ON STRINGS
- SYNTAX:

LOC=MYSTRING.FIND(SUBSTRING) #RETURNS INT FOR INDEX OR -1 IF NOT PRESENT

LOC=MYSTRING.RFIND(SUBSTRING) #SAME AS FIND BUT SEARCHES IN REVERSE ORDER

NEWSTRING=MYSTRING.REPLACE(OLDSUBSTRING,NEWSUBSTRING) #RETURNS NEWSTRING

## FIND EXAMPLES

```
MY_STRING='THIS IS MY STRING'
```

```
MY_INDEX=MY_STRING.FIND('MY')
```

```
S_INDEX=MY_STRING.FIND('S',4)
```

```
LAST_S_INDEX=MY_STRING.RFIND('S')
```

```
FALSE_INDEX=MY_STRING.FIND('S',MY_INDEX,MY_INDEX+2)
```

```
PRINT(MY_INDEX, S_INDEX, LAST_S_INDEX, FALSE_INDEX)
```





## REPLACE EXAMPLE

```
MY_STRING='THIS IS MY STRING'
```

```
NEW_STRING=MY_STRING.REPLACE('MY', 'NOT MY')
```

```
PRINT(NEW_STRING)
```





# STRING COUNT

NUMBER=MY\_STRING.COUNT(SUBSTRING)

COUNTS THE NUMBER OF TIMES THE SUBSTRING OCCURS, RETURNS 0 IF NONE



# SPLIT AND JOIN METHODS

- SYNTAX:

`LIST_OF_SUBSTRINGS=MY_STRING.SPLIT(SEPARATOR_SUBSTRING)`

`COMBINED_STRING=JOINING_STRING.JOIN([STRING1,STRING2,...])`

JOIN CAN BE ACCOMPLISHED WITH THE + OPERATOR AS WELL

## SPLIT EXAMPLE

```
MY_PHONE='865-723-2222'
```

```
MY_LIST=MY_PHONE.SPLIT('-')
```

```
MY_AREA_CODE=MY_LIST[0]
```

```
PRINT(MY_LIST, MY_AREA_CODE)
```



## JOIN EXAMPLE

```
NAME1=INPUT("WHAT IS THE FIRST PERSON'S NAME?")
```

```
NAME2= INPUT("WHAT IS THE SECOND PERSON'S NAME?")
```

```
MY_STRING=' AND '.JOIN([NAME1,NAME2])
```

```
PRINT(MY_STRING)
```



# CHAPTER 9

FILES

## SOME GENERAL COMMENTS

- FILES ARE NORMALLY STORED ON SOME SORT OF LOCAL STORAGE, BUT COULD ALSO BE REMOTE
- PYTHON SUPPORTS TEXT AND BINARY FILES
- FILES ARE FUNDAMENTALLY DESIGNED FOR SEQUENTIAL ACCESS

# FILE MODES

- READ
  - READS FROM THE STORED OBJECT SEQUENTIALLY FROM BEGINNING TO END
- WRITE
  - WRITES TO STORED OBJECT SEQUENTIALLY
- APPEND
  - WRITES BY STARTING AT THE END OF EXISTING STORED OBJECT



# OPEN STATEMENT

- CREATES A FILE OBJECT IN PYTHON
- SYNTAX:

```
FILE=OPEN(FILE_NAME, MODE)
```

MODE CAN BE 'R','W' OR 'A' CAN ADD B TO INDICATE BINARY OPERATION IF OMITTED –READ IS THE DEFAULT

## OPEN EXAMPLE

```
MY_FILE=OPEN('DATA.TXT','R')
```

```
FILE_CONTENTS=MY_FILE.READ() #READS ALL THE DATA INTO A SINGLE STRING
```

```
PRINT(FILE_CONTENTS)
```

## READ AND READLINES(TEXT MODE)

- READ() READS ALL THE CONTENTS INTO A SINGLE STRING  
`ALL_CONTENT=FILE.READ()`
- READLINES() READS THE CONTENT INTO A LIST USING LINE ENDINGS AS A SEPARATOR  
`ALL_CONTENT_LIST=FILE.READLINES()`
- CAN ALSO READ BY USING A FOR LOOP  
FOR LINE IN FILE:  
ACTION BLOCK

## FOR LOOP READING EXAMPLE

```
MY_FILE=OPEN('LISTOFNAMES.TXT') #FORMAT LAST NAME, FIRST NAME
```

```
FOR LINE IN MY_FILE:
```

```
    NAME=LINE.SPLIT(',')
```

```
    PRINT('FIRST NAME IS ',NAME[1])
```

```
    PRINT('LAST NAME IS ',NAME[0])
```

WHY NOT USE READLINES FIRST?



# WRITE STATEMENT

- WRITES CONTENT TO THE FILE
- SYNTAX :  
MY\_FILE.WRITE(OUTPUT\_STRING)

DOES NOT IMMEDIATELY WRITE THE OUTPUT TO DISK – STORES IN INTERNAL BUFFER

BUFFERING CAN BE SPECIFIED IN THE OPEN STATEMENT

FILE\_OBJECT=OPEN(FILE\_NAME,'W',BUFFERING=BUFFER\_SIZE)

# FLUSH AND CLOSE STATEMENTS

- FLUSH FORCES ALL THE CONTENTS OF THE BUFFER TO BE WRITTEN OUT, BUT THE FILE REMAINS OPEN

- SYNTAX:

MYFILE.FLUSH()

- CLOSE FLUSHES THE CONTENTS OF THE BUFFER, AND THE FILE OBJECT IS RELEASE

- SYNTAX:

MYFILE.CLOSE()

- WHY?

# WHY USE FLUSH AND CLOSE

- FLUSH IS USED WHEN ANOTHER PROGRAM OR PROCESS IS READING THE FILE SIMULTANEOUSLY
- CLOSE IS USED FOR:
  - GOOD READABILITY TO INDICATE THE OPERATIONS WITH A GIVEN FILE ARE DONE
  - TO USE FILE OBJECT WITH ANOTHER FILE



## CLOSE EXAMPLE

```
MY_FILES=['DATA1.TXT','DATA2.TXT','DATA3.TXT']
```

```
FOR FILE_NAME IN MY_FILES:
```

```
    F=OPEN(FILE_NAME,'R')
```

```
    PRINT('IN FILE ', FILE_NAME)
```

```
    FOR LINE IN F:
```

```
        PRINT(LINE)
```

```
    F.CLOSE()
```



# BINARY MODE

- WILL READ INTO BYTES TYPE – WILL NOT TREAT CARRIAGE RETURN IN ANY SPECIAL WAY
  - CAN READ ANY FILE TYPE
  - NEED KNOWLEDGE OF CONTENT TO INTERPRET
- WRITE TAKES IN BYTES TYPE AS ARGUMENT

```
IMAGE_OBJECT=OPEN('FLOWER.BMP','RB')
```

```
IMAGE=IMAGE_OBJECT.READ()
```

# UPDATE MODE

- CAN USE THE + IN THE OPEN STATEMENT TO OPEN THE FILE FOR SIMULTANEOUS READING AND WRITING
- 'R+' OPENS EXISTING FILE
- 'W+' OPENS A NEW FILE -- NOT VERY COMMON
- DOING A WRITE IN 'R+' MODE OVERWRITES CONTENTS OF EXISTING FILE – BE VERY CAREFUL!!

## WITH STATEMENT

- ALLOWS CREATION OF AN EXPLICIT BLOCK WHERE A GIVEN FILE IS WORKED WITH
- NO NEED TO EXPLICITLY CLOSE

## WITH EXAMPLE

```
MY_FILES=['DATA1.TXT','DATA2.TXT','DATA3.TXT']
```

```
FOR FILE_NAME IN MY_FILES:
```

```
    WITH OPEN(FILE_NAME,'R') AS F:
```

```
        PRINT('IN FILE ', FILE_NAME)
```

```
    FOR LINE IN F:
```

```
        PRINT(LINE)
```



# CSV MODULE

- USED FOR COMMA SEPARATED TEXT OR OTHER DELIMITERS
- CAN READ CSV FILES AND OTHER TEXT FILES

# CSV FORMAT

BOB, 27, TEXAS

TODD,23,MAINE

JILL,30, NEW MEXICO



# CSV READER

SYNTAX:

```
READER_OBJECT=CSV.READER(FILE, DELIMITER=STRING)
```

READER OBJECT IS A LIST OF LISTS EACH LINE BEING ITS OWN LIST, THE DELIMITER CREATES INDIVIDUAL ITEMS

# CSV READER EXAMPLE

```
IMPORT CSV
```

```
FILE=OPEN('MYTEXT.TXT','R')
```

```
READER=CSV.READER(FILE, DELIMITER=';')
```

```
FOR LINE IN READER:
```

```
    PRINT('THIS LINE HAS ', LEN(LINE), ' ITEMS')
```

```
    PRINT('THE FIRST ITEM IS ', LINE[0])
```

