

# TEC222 Introduction to Programming with Python

## Lecture 3

Strings

Input/Output

Files

# String Functions and Methods

Function or Method	Example	Value	Description
len	len(str1)	6	number of characters in the string
upper	str1.upper()	"PYTHON"	uppercases every alphabetical character
lower	str1.lower()	"python"	lowercases every alphabetical character
count	str1.count('th')	1	number of non-overlapping occurrences of the substring
capitalize	"coDE".capitalize()	"Code"	capitalizes the first letter of the string and lowercases the rest
title	"beN hur".title()	"Ben Hur"	capitalizes the first letter of each word in the string and lowercases the rest
rstrip	"ab ".rstrip()	"ab"	removes spaces from the right side of the string

Table 2.3 String Operations (str1 = "Python")

# [ Chained Methods ]

- Lines can be combined into a single line said to chain the two methods
  - Executed from left to right

```
praise = "Good Doggie".upper()  
numberOfGees = praise.count('G')
```



```
numberOfGees = "Good Doggie".upper().count('G')
```

# [ The input Function ]

- Prompts the user to enter data

```
town = input("Enter the name of your city: ")
```

- User types response, presses ENTER key
- Entry assigned to variable on left

# [ The input Function ]

- Example: Program parses a name

```
fullName = input("Enter a full name: ")  
n = fullName.rfind(" ")  
print("Last name:", fullName[n+1:])  
print("First name(s):", fullName[:n])
```

[Run]

```
Enter a full name: Franklin Delano Roosevelt  
Last name: Roosevelt  
First name(s): Franklin Delano
```

# [ More String Functions ]

- Example 5:

Program  
shows use  
of int, float,  
and eval  
functions

```
print (int ("23"))  
print (float ("23"))  
print (eval ("23"))  
print (eval ("23.5"))  
x = 5  
print (eval ("23 + (2 * x)"))
```

[Run]

23

23.0

23

23.5

33

# String Functions with Numbers

Example	Value	Example	Value
<code>int(4.8)</code>	4	<code>float(4.67)</code>	4.67
<code>int(-4.8)</code>	- 4	<code>float(-4)</code>	-4.0
<code>int(4)</code>	4	<code>float(0)</code>	0.0

- `int` and `float` also work with numbers
- The `str` function converts a number to its string representation

# [ Internal Documentation ]

- Benefits of documentation
- Other people easily understand program.
- You can better understand program when you read it later.
- Long programs are easier to read
  - Purposes of individual pieces can be determined at a glance.



# [ Internal Documentation ]

- Example : Program shows use of documentation.



```
## Break a name into two parts -- the last name and the first names.  
fullName = input("Enter a full name: ")  
n = fullName.rfind(" ") # index of the space preceding the last name  
# Display the desired information.  
print("Last name:", fullName[n+1:])  
print("First name(s):", fullName[:n])
```

# [ Line Continuation ]

- A long statement can be split across two or more lines
  - End each line with backslash character ( \ )
- Alternatively any code enclosed in a pair of parentheses can span multiple lines.
  - This is preferred style for most Python programmers

```
quotation = ("Well written code is its own " +  
             "best documentation.")
```

# Indexing and Slicing Out of Bounds

- Python does not allow out of bounds indexing for individual characters of strings
  - Does allow out of bounds indices for slices
- Given: `str1 = "Python"`
  - Then `print(str1[7])`  `print(str1[-7])` 
  - These are OK

```
str1[-10:10] is "Python"
```

```
str1[-10:3] is "Pyt"
```

```
str1[2:10] is "thon".
```

# [ Exercise ]

- Write a program to request the name of a football team, the number of games won, and the number of games lost as input, and then display the name of the team and the percentage of games won.

Work in Pairs!

[

---

**OUTPUT**

# [ Optional print Argument sep ]

- Consider statement  
`print(value0, value1, ..., valueN)`
- Print function uses string consisting of one space character as separator
- Optionally change the separator to any string we like with the `sep` argument

## Statement

```
print("Hello", "World!", sep="**")  
print("Hello", "World!", sep="")  
print("1", "two", 3, sep="  ")
```

## Outcome

```
Hello**World!  
HelloWorld!  
1    two    3
```

# [ Optional print Argument end ]

- Print statement ends by executing a newline operation.
- Optionally change the ending operation with the end argument

```
print("Hello", end=" ")  
print("World!")
```

[Run]

Hello World!

```
print("Hello", end="")  
print("World!")
```

[Run]

HelloWorld!

# [ Escape Sequences ]

- Short sequences placed in strings
  - Instruct cursor or permit some special characters to be printed.
  - First character is always a backslash (\).
- \t induces a horizontal tab
- \n induces a newline operation



# [ Escape Sequences ]

- Example: Program demonstrates the use of the escape sequences `\t` and `\n`.

```
## Demonstrate use of escape sequences.  
print("01234567890123456")  
print("a\tb\tc")  
print("a\tb\tc".expandtabs(5))  
print("Nudge, \tnudge, \nwink, \twink.".expandtabs(11))
```

[Run]

```
01234567890123456  
a          b          c  
a    b    c  
Nudge,      nudge,  
wink,       wink.
```

# [ Escape Sequences ]

- Backslash also used to treat quotation marks as ordinary characters.
- `\"` causes print function to display double quotation mark
- `\\` causes print function to display single backslash

# [ Justifying Output in a Field ]

- Example : Program demonstrates methods `ljust(n)`, `rjust(n)`, and `center(n)`

```
## Demonstrate justification of output.
print("0123456789012345678901234567")
print("Rank".ljust(5), "Player".ljust(20), "HR".rjust(3), sep="")
print('1'.center(5), "Barry Bonds".ljust(20), "762".rjust(3), sep="")
print('2'.center(5), "Hank Aaron".ljust(20), "755".rjust(3), sep="")
print('3'.center(5), "Babe Ruth".ljust(20), "714".rjust(3), sep="")
```

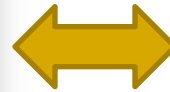
[Run]

```
0123456789012345678901234567
Rank Player           HR
 1  Barry Bonds       762
 2  Hank Aaron        755
 3  Babe Ruth         714
```

# [ Justify Output with format ]

- Given: `str1` is a string and `w` is a field width

```
print("{0:<ws}".format(str1))  
print("{0:^ws}".format(str1))  
print("{0:>ws}".format(str1))
```

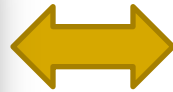


```
print(str1.ljust(w))  
print(str1.center(w))  
print(str1.rjust(w))
```

# [ Justify Output with format ]

- Given: num is a number and w is a field width

```
print("{0:<wn}".format(num))  
print("{0:^wn}".format(num))  
print("{0:>wn}".format(num))
```



```
print(str(num).ljust(w))  
print(str(num).center(w))  
print(str(num).rjust(w))
```

# [ Justify Output with format ]

- Example 3: Program illustrates formatting

```
## Demonstrate justification of output.  
print("0123456789012345678901234567")  
print("{0:^5s}{1:<20s}{2:>3s}".format("Rank", "Player", "HR"))  
print("{0:^5n}{1:<20s}{2:>3n}".format(1, "Barry Bonds", 762))  
print("{0:^5n}{1:<20s}{2:>3n}".format(2, "Hank Aaron", 755))  
print("{0:^5n}{1:<20s}{2:>3n}".format(3, "Babe Ruth", 714))
```

[Run]

```
0123456789012345678901234567  
Rank Player HR  
 1 Barry Bonds 762  
 2 Hank Aaron 755  
 3 Babe Ruth 714
```

# [ Justify Output with format ]

- Table 2.4 Demonstrate number formatting.

Statement	Outcome	Comment
<code>print("{0:10d}".format(12345678))</code>	12345678	number is an integer
<code>print("{0:10,d}".format(12345678))</code>	12,345,678	thousands separators added
<code>print("{0:10.2f}".format(1234.5678))</code>	1234.57	rounded
<code>print("{0:10,.2f}".format(1234.5678))</code>	1,234.57	rounded and separators added
<code>print("{0:10,.3f}".format(1234.5678))</code>	1,234.568	rounded and separators added
<code>print("{0:10.2%}".format(12.345678))</code>	1234.57%	changed to % and rounded
<code>print("{0:10,.3%}".format(12.34567))</code>	1,234.568%	%, rounded, separators

# [ Justify Output with format ]

- Example: Program formatting with curly brackets

```
## Demonstrate use of the format method.  
print("The area of {0:s} is {1:,d} square miles.".format("Texas", 268820))  
str1 = "The population of {0:s} is {1:.2%} of the U.S. population."  
print(str1.format("Texas", 26448000 / 309000000))
```

[Run]

```
The area of Texas is 268,820 square miles.  
The population of Texas is 8.56% of the U.S. population.
```



# Exercise

## ■ What is printed?

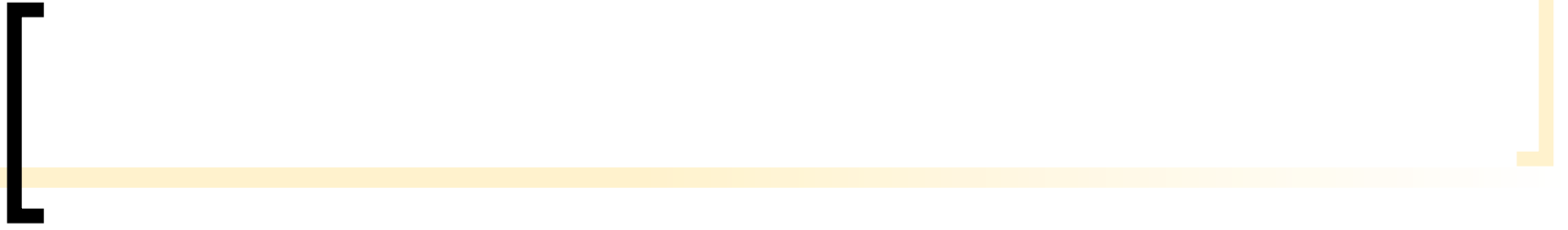
```
print("{0:14s}{1:s}".format("Major", "Percent of Students"))
print("{0:14s}{1:10.1%}".format("Biology", .062))
print("{0:14s}{1:10.1%}".format("Psychology", .054))
print("{0:14s}{1:10.1%}".format("Nursing", .047))
```

```
-----
print("When nothing goes {0:s} go {1:s}.".format("right", "left"))
```

```
-----
print("Plan {0:s}, code {1:s}.".format("first", "later"))
```

```
-----
print("{0:s} are the {1:s} of {0:s}r own destiny".format("you","creator"))
```

```
-----
print("And now for {0:s} completely {1:s}.".format("something", "different"))
```



# LISTS

# [ The list Object ]

- A list is an ordered sequence of Python objects
  - Objects can be of any type
  - Objects do not have to all be the same type.
  - Constructed by writing items enclosed in square brackets ... items separated by commas.

```
team = ["Seahawks", 2014, "CenturyLink Field"]  
nums = [5, 10, 4, 5]  
words = ["spam", "ni"]
```

# [ The list Object ]

- Table 2.5 List operations (The lists team, num, and words given in previous slide)

Function or Method	Example	Value	Description
len	len(words)	2	number of items in list
max	max(numbers)	10	greatest (items must have same type)
min	min(numbers)	4	least (items must have same type)
sum	sum(nums)	42	total (items must be numbers)
count	nums.count(5)	2	number of occurrences of an object
index	nums.index(4)	2	index of first occurrence of an object
reverse	words.reverse()	["ni", "spam"]	reverses the order of the items

# [ The list Object ]

- Table 2.5 List operations (The lists *team*, *num*, and *words* given in previous slide)

reverse	<code>words.reverse()</code>	<code>["ni", "spam"]</code>	reverses the order of the items
clear	<code>team.clear()</code>	<code>[]</code>	<code>[]</code> is the empty list
append	<code>nums.append(7)</code>	<code>[5, 10, 4, 5, 7]</code>	inserts object at end of list
extend	<code>nums.extend([1, 2])</code>	<code>[5, 10, 4, 5, 1, 2]</code>	inserts new list's items at end of list
del	<code>del team[-1]</code>	<code>["Seahawks", 2014]</code>	removes item with stated index
remove	<code>nums.remove(5)</code>	<code>[10, 4, 5]</code>	removes first occurrence of an object
insert	<code>nums.insert(1, "wink")</code>	<code>["spam", "wink", "ni"]</code>	insert new item before item of given index
+	<code>['a', 1] + [2, 'b']</code>	<code>['a', 1, 2, 'b']</code>	concatenation; same as <code>['a', 1].extend([2, 'b'])</code>
*	<code>[0] * 3</code>	<code>[0, 0, 0]</code>	list repetition

# [ The list Object ]

- Example: Program requests five grades as input, displays average after dropping two lowest grades

```
## Calculate average of grades
grades = []    # Create the variable grades and assign it the empty list.
num = float(input("Enter the first grade: "))
grades.append(num)
num = float(input("Enter the second grade: "))
grades.append(num)
num = float(input("Enter the third grade: "))
grades.append(num)
num = float(input("Enter the fourth grade: "))
grades.append(num)
num = float(input("Enter the fifth grade: "))
grades.append(num)
```

# [ The list Object ]

- Program requests five grades as input, displays average after dropping two lowest grades

```
grades.append(num)
minimumGrade = min(grades)
grades.remove(minimumGrade)
minimumGrade = min(grades)
grades.remove(minimumGrade)
average = sum(grades) / len(grades)
print("Average Grade: {0:.2f}".format(average))
```

[Run]

```
Enter the first grade: 89
Enter the second grade: 77
Enter the third grade: 82
Enter the fourth grade: 95
Enter the fifth grade: 81
Average Grade: 88.67
```

# [ Slices ]

- A slice of a list is a sublist specified with colon notation
  - Analogous to a slice of a string
- Table 2.6 Meanings of slice notations

Slice Notation	Meaning
<code>list1[m:n]</code>	list consisting of the items of <i>list1</i> having indices <i>m</i> through <i>n</i> – 1
<code>list1[:]</code>	a new list containing the same items as <i>list1</i>
<code>list1[m:]</code>	list consisting of the items of <i>list1</i> from <code>list1[m]</code> through the end of <i>list1</i>
<code>list1[:m]</code>	list consisting of the items of <i>list1</i> from the beginning of <i>list1</i> to the element having index <i>m</i> – 1



# [ Slices ]

- Table 2.7 Examples of slices where `list1 = ['a', 'b', 'c', 'd', 'e', 'f']`.

Example	Value
<code>list1[1:3]</code>	<code>['b', 'c']</code>
<code>list1[-4:-2]</code>	<code>['c', 'd']</code>
<code>list1[:4]</code>	<code>['a', 'b', 'c', 'd']</code>
<code>list1[4:]</code>	<code>['e', 'f']</code>
<code>list1[:]</code>	<code>['a', 'b', 'c', 'd', 'e', 'f']</code>
<code>del list1[1:3]</code>	<code>['a', 'd', 'e', 'f']</code>
<code>list1[2:len(list1)]</code>	<code>['c', 'd', 'e', 'f']</code>
<code>(list1[1:3])[1]</code>	<code>'c'</code> (This expression is usually written as <code>list1[1:3][1]</code> )
<code>list1[3:2]</code>	<code>[]</code> , the list having no items; that is, the empty list

# [ The split and join Methods ]

- Split method turns single string into list of substrings
- Join method turns a list of strings into a single string.
- Notice that these methods are inverses of each other

# [ The split and join Methods ]

- Example 2: These statements each display list ['a', 'b', 'c'].

```
print("a,b,c".split(','))  
print("a**b**c".split '**'))  
print("a\nb\nc".split())  
print("a b c".split())
```

# [ The split and join Methods ]

- Example 3: Program shows how join method used to display items from list of strings.

```
line = ["To", "be", "or", "not", "to", "be."]
print(" ".join(line))
krispies = ["Snap", "Crackle", "Pop"]
print(", ".join(krispies))
```

[Run]

```
To be or not to be.
Snap, Crackle, Pop
```

# [ Text Files ]

---

- Text file is a simple file consisting of lines of text with no formatting
  - Text file can be created with any word processor
  - Python program can access the values

# [Text Files]

## ■ Given

```
infile = open("Data.txt", 'r')  
listName = [line.rstrip() for line in infile]  
infile.close()
```

- Program opens for reading a file of text
- Strips newline characters
- Characters placed into a list

# [Text Files]

## ■ Furthermore

```
infile = open("Data.txt", 'r')  
listName = [eval(line) for line in infile]  
infile.close()
```

- Suppose data is all numbers,
- Previous code produces list of strings, each a number
- Place the numbers into a list with this code

# [ Exercise ]

- Write a program that requests a two-part name and then displays the name in the form "lastName, firstName".

```
Enter a 2-part name: John Doe  
Revised form: Doe, John
```



# [ Questions ? ]

