CS65: Introduction to Computer Science

Tuple
String Methods
String Formatting

Md Alimoor Reza
Assistant Professor of Computer Science

Mid Semester Evaluation

https://tinyurl.com/v5cxe4d7

Topic

- Recap (from last lecture): Dictionary
- Tuple

- Strings
  - useful methods

- Strings formating
  - with % operator
  - with **.format()**

Review: Dictionary

- Only one entry per key is allowed! When there is a duplicate, the last entry wins

```
my_dict = {'one': 1, 'two': 2, 'one': 3}

print(my_dict['one'])
```

- Lists are not allowed as ***keys***
- No restrictions on ***values***
- Dictionaries do not keep order
- Keys must be **unique** and **immutable**

Tuple: another type of a sequence

- Cannot change Tuple's items after creation (**immutability**)
- Items are accessed by indices
  - similar to other two sequences **List** and **String**

Sequence	Example	Syntax	Accessing
String	my_str = "My name is walle"	within enclosing quotation marks, ie, " " or ' '	my_str[0] my_str[1]
List	my_list = [1, 2, "a", "abs"]	within enclosing brackets **[ ]** and separated by commas	my_list[0] my_list[1]
Tuple	my_tuple = (1, 2, "a", "abs")	Within enclosing parenthesis **( )** and separated by commas	my_tuple[0] my_tuple[1]

Mutable Property of List

```
# ---------- mutability of List ------------

my_list = [1, 2, "a", "abs"]

for i in range(len(my_list)):
    print(my_list[i])


# trying to update a location with a new value

my_list[1] = 3

print("modified value of list ", my_list[1])
```

```
>>> %Run lec14demo.py

  1
  2
  a
  abs
  modified value of list  3
```

Immutable Property of Tuple

```
# ---------- immutability of Tuple -------------
my_tuple = (1, 2, "a", "abs")

for i in range(len(my_tuple)):
    print(my_tuple[i])


# trying to update a location with a new value

my_tuple[1] = 3

print("modified value of tuple ", my_tuple[1])
```

```
1
2
a
abs
Traceback (most recent call last):
  File "/Users/reza/Class_and_Research/drake_teaching/CS65/c
>
    my_tuple[1] = 3
TypeError: 'tuple' object does not support item assignment
```

Tuple

- **Tuple examples**

```
# tuple examples

tup1 = ()

tup2 = (1,)          # one-tuple needs a comma in Python

tup3 = ("Georg Cantor", "Bertrand Russell", "Kurt Godel")

tup4 = (True, False, True, False)

tup5 = ([1, 2, 3], [4, 5, 6])

tup6 = ((1, 2, 3), (4, 5, 6))
```

- Exercise: Try the examples above in Thonny. Find the items you can modify and which the ones you cannot

Exercise

```
tup1 = ()
print(len(tup1))            # what is the length?

tup2 = (1,)                 # one-tuple needs a comma


tup3 = ("Georg Cantor", "Bertrand Russell", "Kurt Godel")
for i in range(len(tup3)): # check out the values
    print(tup3[i])


tup4 = (True, False, True, False)
print(tup4[0])


# Tuple of list
tup5       = ([1, 2, 3], [4, 5, 6])
tup5[0][0] =  10            # modify the value of the first entry of the first inner tuple
print(tup5)
tup5[0]    = [10, 20, 30]  # modify the value of the first entry



# Tuple of Tuples
tup6       = ((1, 2, 3), (4, 5, 6))
tup6[0][0] =  10            # modify the first entry of the first inner tuple
print(tup6)
tup6[0]    = (10, 20, 300) # modify the first entry
```

Topic

- Recap (from last lecture): Dictionary
- Tuple

- Strings
  - useful methods

- Strings formating
  - with % operator
  - with **.format()**

Useful string methods

- **Syntax:** string_expression.method_name($argm_1$, $argm_2$, $argm_n$)

method	purpose	returned value
s.upper() s.lower()	converts letters to upper or lower case	modified copy of s
s.startswith(svar[,start[,stop]]) s.endswith(svar[,start[,stop]])	is svar a prefix/suffix of s?	Boolean value
s.join(iterable)	concatenates strings from iterable, with copies of string s inbetween them	string result of all those concatenations/ interspersings
s.split(sep)	get list of strings obtained by splitting s into parts at each occurrence of sep	list of strings from between occurrences of sep
s.replace(old, new[,count])	replace all (or count) occurrences of old str with new str.	string with replacements performed

Useful String Methods: .upper()

- **Syntax:** string_expression.method_name()

  my_str.upper( )

  my_str.lower( )

```
#-------------------------------------------------------------
#                    .upper() or .lower()
# ------------------------------------------------------------
my_str       = "drake university"
my_str_upper = my_str.upper()

print("upper(): ", my_str_upper)
print("lower(): ", "HELLO".lower())
```

```
upper():  DRAKE UNIVERSITY
lower():  hello
```

Useful String Methods: .split()

- **Syntax:** string_expression.method_name(*argm1*)

  my_str.upper (*separator*)

```
# ------------------------------------------------------------
#                    .split() method
# ------------------------------------------------------------
my_str         = "computer,science,department"
splitted_items = my_str.split(',')
for val in splitted_items:
    print("splitted strings are: ", val)
```

```
splitted strings are:  computer
splitted strings are:  science
splitted strings are:  department
```

Useful String Methods: .replace()

- **Syntax:** string_expression.method_name($argm_1$, $argm_2$)

  my_str.replace (old_{str} , new_{str})

```
#-------------------------------------------------------------
#                    .replace()
# ------------------------------------------------------------
my_str      = "A brown quick fox jump over the lazy dog"
new_str     = my_str.replace("lazy", "tired")

print("old : ", my_str)
print("new : ", new_str)
```

```
old :  A brown quick fox jump over the lazy dog
new :  A brown quick fox jump over the tired dog
```

Useful String Methods: .replace()

- **Syntax:** string_expression.method_name($argm_1$, $argm_2$, $argm_3$)

my_str.replace (old_{str} , new_{str} , *how_many_times*)

```
my_str = "A brown quick fox jump over the lazy lazy lazy dog"
new_str = my_str.replace("lazy", "tired", 2)

print("old : ", my_str)
print("new : ", new_str)
```

```
old :  A brown quick fox jump over the lazy lazy lazy dog
new :  A brown quick fox jump over the tired tired lazy dog
```

Useful String Methods: .find()

- **Syntax:** string_expression.method_name($argm_1$)

  my_str.find ($str_{you\_are\_looking\_for}$)

```
#-------------------------------------------------------------
#                    .find()
# ------------------------------------------------------------
my_str      = "A brown quick fox jump over the lazy dog"
position    = my_str.find("lazy")

print("string : ", my_str)
print("lazy at position {}", position)
```

```
string :  A brown quick fox jump over the lazy dog
lazy at position {} 32
```

Topic

- Recap (from last lecture): Dictionary
- Tuple

- Strings
  - useful methods

- Strings formating
  - with **%** operator
  - with **.format()**

String Formating

- Two popular approaches:
  - Percent operator **%**
  - **format()** method

String Formating: Percent Operator %

- Percent operator %
  - describe pattern of string with placeholder with % operator, then supply all substitutions at once
  - string_expression **%** (*tuple_item*$_1$, *tuple_item*$_2$, *tuple_item*$_n$)

```
>>> "A week has %d days, and a year has %d months"%(7, 12)
'A week has 7 days, and a year has 12 months'
```

String formating: % operator

format pattern	style of output	accepted input
%d	integer	integers, floats
%f	float	integers, floats
%g	float (scientific notation)	integers, floats – but it prefers scientific notation representation
%s	string	anything (calls str())
%%	the '%' character	none – just represents the % symbol

```
>>> "%s loves to sleep %f hours a day" % ("Reza", 7.55)
'Reza loves to sleep 7.550000 hours a day'
```

String formating: % operator

format pattern	style of output	accepted input
%d	integer	integers, floats
%f	float	integers, floats
%g	float (scientific notation)	integers, floats – but it prefers scientific notation representation
%s	string	anything (calls str())
%%	the '%' character	none – just represents the % symbol

```
#-------------------------------------------------------------------------
#                    string formating with % operator
#-------------------------------------------------------------------------
str1 = "A week has %d days, and a year has %d months" % (7,      12)
str2 = "%s loves to sleep %f hours a day"             % ("Reza", 7.55)
str3 = "Speed of light is %g meters/second"           % (2.998e+8)
str4 = "Speed of light is %d meters/second"           % (2.998e+8)

print(str1)
print(str2)
print(str3)
print(str4)
```

```
A week has 7 days, and a year has 12 months
Reza loves to sleep 7.550000 hours a day
Speed of light is 2.998e+08 meters/second
Speed of light is 299800000 meters/second
```

Scientific number

Integer number

String formating: % operator

purpose	examples	results
state exact # columns after decimal point (%f)	`"%.2f" % (2/3)` `"%.0f" % 15.5`	`'0.67'` `'16'`

```
#-------------------------------------------------------------------------
#                    string formating with % operator
#  showing EXACT number of digits after the decimal point (floating numbers)
#-------------------------------------------------------------------------
print("%.0f" % (2.555))
print("%.1f" % (2.555))
print("%.2f" % (2.555))
print("%.3f" % (2.555))
```

col_1	col_2	col_3	col_4	col_5
3				
2	.	6		
2	.	5	6	
2	.	5	5	5

```
3
2.6
2.56
2.555
```

String formating: % operator

purpose	examples	results
state min. # columns	"%4d" % 30	' 30'
for entire thing	"%3d" % 1234	'1234'

```
#-------------------------------------------------------------------------
#                  string formating with % operator
#        showing MINIMUM number of columns for the entire thing
#-------------------------------------------------------------------------
print("%1d" % (3))
print("%2d" % (3))
print("%3d" % (3))
print("%4d" % (3))
print("%4d" % (1234))
```

col_1	col_2	col_3	col_4
3			
	3		
		3	
			3
1	2	3	4

```
3
  3
    3
      3
1234
```

String formating: % operator

purpose	examples	results
state min. # columns	"%4d" % 30	' 30'
for entire thing	"%3d" % 1234	'1234'

```
#-------------------------------------------------------------------------------
#                  string formating with % operator
#        showing MINIMUM number of columns for the entire thing
#            filling in (the leading empty spaces) with ZEROS
#-------------------------------------------------------------------------------
print("%03d" % (1))
print("%04d" % (2))
print("%05d" % (3))
```

col_1	col_2	col_3	col_4	col_5
0	0	1		
0	0	0	2	
0	0	0	0	3

```
001
0002
00003
```

String formating: % operator

```
#-------------------------------------------------------------------------------
#                    string formating with % operator
#                        floating point numbers
#          showing MINIMUM number of columns for the entire thing
#                                  +
#          showing EXACT number of digits after decimal point
#-------------------------------------------------------------------------------
print("%04.2f" % (2.555))
print("%05.2f" % (2.555))
print("%06.2f" % (2.555))
print("%07.2f" % (2.555))
print("%08.2f" % (2.555))
```

col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8
2	.	5	6				
0	2	.	5	6			
0	0	2	.	5	6		
0	0	0	2	.	5	6	
0	0	0	0	2	.	5	6

```
2.56
02.56
002.56
0002.56
00002.56
```

String Formating

- Two popular approaches:
  - Percent operator **%**
  - **format()** method

String formating with .format() method

- **.format()** method
  - include {}'s as placeholders in string, put style rules inside
  - provide the substitutions as arguments to .format() method
  - {} divide by {} is {} **%.format(** $args_1$, $args_2$ **,** $args_n$**)**

String formating with .format() method

```
#-------------------------------------------------------------------------
#                    string formating with .format() method
#-------------------------------------------------------------------------

my_str = "{:06.2f}".format(12.3456)
print(my_str, "\n")

str0 = "{} by {} is {:.2f}".format(2.5, 3, 0.8333333)
print(str0)
```

col_1	col_2	col_3	col_4	col_5	col_6
0	1	2	.	3	5

```
012.35

2.5 by 3 is 0.83
```

String formating with .format() method

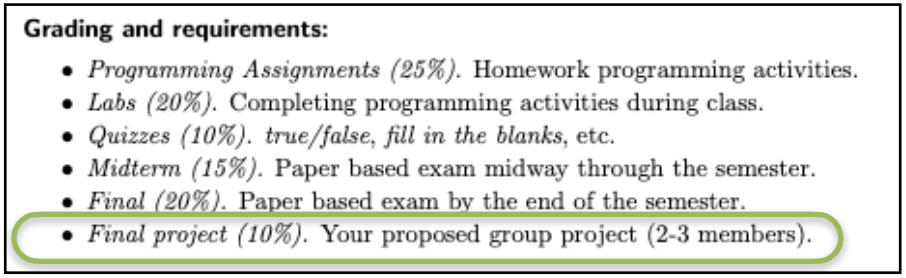
```
#-------------------------------------------------------------------------
#                    string formating with .format() method
#-------------------------------------------------------------------------

str1 = "{:>8} ".format("yo") # right align
str2 = "{:<8} ".format("yo") # left align
str3 = "{:^8} ".format("yo") # center align
str4 = "{:*^8}".format("yo") # center align + fill with *
str5 = "{:@^8}".format("yo") # center align + fill with @
str6 = "{:2^8}".format("yo") # center align + fill with 2

print(str1, "\n")
print(str2, "\n")
print(str3, "\n")
print(str4, "\n")
print(str5, "\n")
print(str6, "\n")
```

col_1	col_2	col_3	col_4	col_5	col_6	col_7	col_8
						y	o
y	o						
			y	o			
*	*	*	y	o	*	*	*
@	@	@	y	o	@	@	@
2	2	2	y	o	2	2	2

```
      yo

yo

   yo

***yo***

@@@yo@@@

222yo222
```

Final Project

- Final project will be due before the end of the semester



Grading and requirements:

- *Programming Assignments (25%).* Homework programming activities.
- *Labs (20%).* Completing programming activities during class.
- *Quizzes (10%). true/false, fill in the blanks,* etc.
- *Midterm (15%).* Paper based exam midway through the semester.
- *Final (20%).* Paper based exam by the end of the semester.
- *Final project (10%).* Your proposed group project (2-3 members).

- The project will also include a final demonstration of the project: 1st week of May (during class time)

- The project proposal is due **next Thursday 07/07**

Final Project Proposal

- This is an opportunity for you to explore a project of your own choosing that uses Python programming

- Submit a **1/2 to 1 page** written project proposal with a storyboard for your final project.
  - A few rough sketches of what your program will look like to help guide the development
  - Your storyboard should be a graphical description of the "flow" of your project
    - you can use Powerpoint (windows) or Keynote (OS X) to prepare the flowchart or other graphical description)

Final Project Proposal

- Your project proposal should either be a Word processing document or in a PDF file format and it should include the following information:

1. Your name or team members' names

2. The idea (the game, simulation, visualization, etc. you plan to implement for your final project)

3. Any datasets you plan to use in your project (if applicable)

4. A development plan:

- What will you do first, second, third, etc.
- What functions will you use or develop?

Final Project Proposal

- Submit the project proposal on Blackboard (10 points)

- One PDF is sufficient for your group as long as it has the names of the members

- The project proposal is due **next Thursday 04/07/2022**

Final Project Ideas

- Text-based games: dice game, tic-tac-toe, etc
- Data analytics: explore any dataset, visualize properties, compute statistics, etc
- Graphics game
- Simulation
- Or anything else of your choice

Final Project Submission

- Submit the project code and report/readme file on Blackboard (100 points)

- For full credit, you should plan on investing several hours
  - You will be asked during your presentation

- The final project is due on **May 12, 2022**
  - You have more than a month so plan accordingly
  - Sooner the better