CS112
Introduction to Python Programming

Session 04: Dictionaries

Shengwei Hou
Ph.D., Assistant Professor
Department of Ocean Science and Engineering

Fall 2022

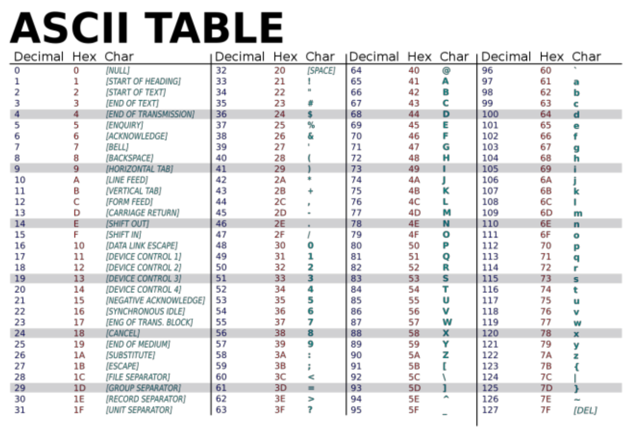
SUSTech Southern University of Science and Technology

明德求是 日新自强
VIRTUE | TRUTH | ADVANCE

Contents

String review

- String is a part of Python's core data structures;
- Strings are **immutable**, it cannot be modified. The characters in a string cannot be changed once a string value is assigned to string variable;
- Strings are created by enclosing a sequence of characters or numbers within a pair of single or double quotes.
- Strings can be concatenated using the "+" operator;
- Strings can also be repeated using the "*" operator;
- `str()` function converts the other data type to a String data type;
- `len()` gives the number of characters in a string;
- `min()` and `max()` return a character having the highest or lowest ASCII value;
- Square brackets are used to perform indexing in a string;
- **Python's index starts from zero.** Each of the string's character corresponds to an index number starting from 0;
- `in` and `not in`: check for the presence of a string in another string ;
- Python compares strings based on the ASCII value of the characters using >, <, <=, >=, ==, !=



ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

String review

SUSTech Southern University of Science and Technology

- String slicing returns a sequence of characters beginning at start and extending up to but not including end; the index numbers are separated by a colon;
- A third argument called **step** can be specified along with the **start** and **end** index numbers to specify steps in slicing; the default value of step is one;
- Strings can be joined with the `.join()` method;
- Strings can be split with the `.split()` method;
- A given string is split into list of strings based on the specified separator;
- If the separator is not specified, then whitespace is considered as the delimiter;
- Some basic String methods:

  `.isalpha()`, `.isdigit()`, `.islower()`, `.isupper()`, `.upper()`, `.lower()`,

  `.capitalize()`, `.find()`, `.count()`, `.swapcase()`
- The `print` function prints everything as strings onto the console;
- Use `str.format()` method if you need to insert the value of a variable, expression or an object into another string.

String review

- f-string format: `f"string_statements {variable_name [:{width}.{precision}]}"`
- Using *variable_name* along with either *width* or *precision* values should be separated by a **:**
- *Precision* refers to the total number of digits that will be displayed in a number
- By default, strings are left-justified and numbers are right-justified
- Escape sequences are a combination of a backslash (\) followed by either a letter or a combination of letters and digits ;
- A raw string is created by prefixing the character *r* to the string;
- A raw string ignores all types of formatting within a string including the escape characters.

```
In [7]: width = 10
In [8]: precision = 5
In [9]: value = 12.34567
In [10]: f'result: {value:{width}.{precision}}'
Out[10]: 'result:     12.346'
In [11]: precision = 3
In [12]: f'result: {value:{width}.{precision}}'
Out[12]: 'result:       12.3'
In [13]: width = 6
In [14]: f'result: {value:{width}.{precision}}'
Out[14]: 'result:   12.3'
```

\\	Inserts a Backslash character in the string
\'	Inserts a Single Quote character in the string
\"	Inserts a Double Quote character in the string
\n	Inserts a New Line in the string
\t	Inserts a Tab in the string
\b	Inserts a Backspace in the string

```
In [15]: print(r"he asked, \"What\'s the best way to code the snake game?\" She answered, \"In *python* script\"")
he asked, \"What\'s the best way to code the snake game?\" She answered, \"In *python* script\"
```

List review

SUSTech Southern University of Science and Technology

- List is a container that holds a number of items;
- Lists are defined by a pair of square brackets with individual elements (can be of mixed type) separated by commas;
- Lists are **mutable** in nature as the list items can be modified after creation;
- Lists can be concatenated using the `+` sign, and the `*` operator is used to create a repeated sequence of list ;
- `range()` creates a uniformly spaced sequence of integers. Its general form is `range([start,] stop[, step])`;
- Individual elements of a list can be accessed using the variable name for the list with an integer in square brackets (Python's index starts from zero) ;
- The last element can also be by index -1.

```
>>> b = [7., "girls & boys", 2+0j, 3.14159, 21]
>>> b[1:4]
['girls & boys', (2+0j), 3.14159]
>>> b[2:]
[(2+0j), 3.14159, 21]
>>> b[:3]
[7.0, 'girls & boys', (2+0j)]
>>> b[:]
[7.0, 'girls & boys', (2+0j), 3.14159, 21]
>>> b[1:-1]
['girls & boys', (2+0j), 3.14159]
```

```
>>> b[::2]
[7.0, (2+0j), 21]
>>> b[1::2]
['girls & boys', 3.14159]
>>> b[::-1]
[21, 3.14159, (2+0j), 'girls & boys', 7.0]
>>> b[1:-1]
['girls & boys', (2+0j), 3.14159]
>>> b[::2]
[7.0, (2+0j), 21]
>>> b[1::2]
['girls & boys', 3.14159]
>>> b[::-1]
[21, 3.14159, (2+0j), 'girls & boys', 7.0]
```

`[start:end:stepsize]`

The default step size is 1

List review

- Assigning an existing list variable to a new variable does not create a new copy of the existing list items;
- Instead, assignment makes both the variable names point to the same list in memory;
- The `sorted()` function returns a modified copy of the list while without modifying the original list, the list is sorted based on the ASCII value;
- The `.append()` method adds a single item to the end of the list;
- The `.count()` method counts the number of times the item has occurred in the list and returns it;
- The `.insert()` method inserts the item at the given index, shifting items to the right;
- The list size changes dynamically whenever you add or remove the items;
- `.index()` method returns the index for the given item from the start of the list;
- `.copy()` method creates a new copy of the existing list;
- The `.remove()` method searches for the first instance of the given item in the list and removes it;
- The `.pop()` method removes and returns the item at the given index, the last item will be returned if no index is given;
- `del` statement removes an item from a list based on its index;
- The difference between `del` and `.pop()` is that `del` does not return any value while `.pop()` returns a value;
- The `del` statement can also be used to remove slices from a list or clear the entire list;
- Multidimensional lists are lists of lists, or nested list.

SUSTech Southern University of Science and Technology

Dictionaries

Dictionary

- A Python list is a collection of Python objects indexed by an ordered sequence of integers starting from zero;
- A dictionary is also a collection of Python objects, **a part of core Python**, but one that is indexed by strings or numbers (not necessarily integers and not in any particular order) or even tuples!
- A dictionary is a collection of an unordered set of **`key:value`** pairs, with the requirement that **the keys must be unique within a dictionary;**
- Dictionaries are created using **curly brackets `{}`**:

```
dictionary_name = {key_1:value_1, key_2:value_2,
key_3:value_3, ………,key_n:value_n}
```

- Dictionaries are sometimes found in other languages as “associative memories” or “associative arrays.”

Dictionary format

- A dictionary of room numbers indexed by the name of the person who occupies each room:

```
>>> room = {"Emma":309, "Jake":582, "Olivia":764}
>>> type(room)
<class 'dict'>
```

- The entries in a dictionary are separated by **commas**;
- Each entry consists of a key and a value; each key and its value are separated by a **colon** ;
- **Dictionaries are indexed by keys**, and the entries can be accessed by keys: `dictionary_name[key]`

```
>>> room["Olivia"]
764
```

keys and values

- The key can be a string, a number, or even a tuple, but it cannot be a list (mutable!):

```
>>> weird = {"tank":52, 846:"horse", 'bones': [23, 'fox', 'grass'],  'phrase':
'I am here'}
>>> weird["tank"]
52
>>> weird[846]
'horse'
>>> weird["bones"]
[23, 'fox', 'grass']
```

- Dictionary keys are case sensitive, and must be ~~immutable~~!
- Duplicate keys are not allowed in the dictionary;
- A value in the dictionary can be of any data type, including string, number, list, or dictionary itself;
- The keys and their associated values can be of different types.

Dictionary creation

SUSTech Southern University of Science and Technology

- Dictionaries can be built up and added to in a straightforward manner:

```
>>> d = {}
>>> d["last name"] = "Alberts"
>>> d["first name"] = "Marie"
>>> d["birthday"] = "January 27"
>>> d
{'last name': 'Alberts', 'first name': 'Marie', 'birthday': 'January
27'}
```

Keys	Values
`'last name'`	`'Alberts'`
`'first name'`	`'Marie'`
`'birthday'`	`'January 27'`

Starting from Python 3.6, the output of dictionary statement is ordered `key:value` *pairs with the sequence of insertion*

Dictionary

- In dictionaries, the order of `key:value` pairs does not matter:

```
 >>> d_new = {'birthday': 'January 27', 'first name': 'Marie', 'last name':
'Alberts'}
 >>> d == d_new
 True
```

- Slicing in dictionaries is not allowed since they are not ordered like lists;

- A list of all keys or values of a dictionary can be obtained by the dictionary name followed by `.keys()` or `.values()` :

```
>>> d.keys()
dict_keys(['last name', 'first name', 'birthday‘])
>>> d.values()
dict_values(['Alberts', 'Marie', 'January 27'])
```

Dictionary modification

- The syntax for modifying the value of an existing key or for adding a new `key:value` pair to a dictionary is:

```
dictionary_name[key] = value
```

- If the key is already in the dictionary, its value will be updated with the new value; if the key is not present, the new `key:value` pair will be added to the dictionary.

```
>>> d = {'last name': 'Alberts', 'first name': 'Marie', 'birthday':
'January 27'}
>>> d["last name"] = "Johnson"
>>> d
{'last name': 'Johnson', 'first name': 'Marie', 'birthday': 'January 27'}
>>> d["phone number"] = 5053101021
>>> d
{'last name': 'Johnson', 'first name': 'Marie', 'birthday': 'January 27',
'phone number': 505310100}
```

Dictionaries are mutable

Dictionary

- The presence of a key in the dictionary using `in` and `not in` membership operators:

```
>>> "last name" in d
True
>>> "address" in d
False
```

- The built-in `dict()` function is used to create dictionary.

```
>>> numbers = dict(one=1, two=2, three=3)
>>> numbers
{'one': 1, 'two': 2, 'three': 3}
>>> new_dict = dict()
>>> new_dict
{}
```

Built-in functions

- The `len()` function returns the number of elements, or `(key:value)` pairs in a dictionary:
- The `sorted()` function by default returns a list of key elements, which are sorted based on dictionary keys:

```
In [4]: d
Out[4]:
{'last name': 'Johnson',
 'first name': 'Marie',
 'birthday': 'January 27',
 'phone number': 5053101021}
In [5]: len(d)
Out[5]: 4
```

```
In [6]: sorted(d)
Out[6]: ['birthday', 'first name', 'last name', 'phone number']
In [8]: sorted(d, reverse=True)
Out[8]: ['phone number', 'last name', 'first name', 'birthday']
In [9]: sorted(d.items())
Out[9]:
[('birthday', 'January 27'),
 ('first name', 'Marie'),
 ('last name', 'Johnson'),
 ('phone number', 5053101021)]
In [10]: sorted(d.items(), reverse=True)
Out[10]:
[('phone number', 5053101021),
 ('last name', 'Johnson'),
 ('first name', 'Marie'),
 ('birthday', 'January 27')]
```

Dictionary methods

- Dictionary allows you to store data in `key:value` format without depending on indexing:

```
>>> dir(dict)
['__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
'__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear',
'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem',
'setdefault', 'update', 'values']
```

Dictionary methods

- `.fromkeys()` creates a new dictionary from the given sequence of elements with a value provided by the user:

```
>>> numbers = {'one': 1, 'two': 2, 'three': 3}
>>> numbers_fromkeys = numbers.fromkeys(numbers)
>>> numbers_fromkeys
{'one': None, 'two': None, 'three': None}
>>> numbers_fromkeys = numbers.fromkeys(numbers, "nums")
>>> numbers_fromkeys
{'one': 'nums', 'two': 'nums', 'three': 'nums'}
```

Dictionary methods

- `.get()` returns the value associated with the specified key in the dictionary. If the key is not present then it returns the default value. If default is not given, it defaults to None, so that this method never raises a **`KeyError`**

```
>>> numbers = {'one': 1, 'two': 2, 'three': 3}
>>> numbers.get("three")
3
>>> print(numbers.get("four"))
None
>>> numbers.get("four", 4)
4
```

```
In [1]: numbers['four']
KeyError          Traceback (most recent call last)
<ipython-input-6-175332202980> in <module>
----> 1 numbers['four']

KeyError: 'four'

In [2]: numbers.get?
Signature: numbers.get(key, default=None, /)
Docstring: Return the value for key if key is in the dictionary, else default.
Type:      builtin_function_or_method
```

Dictionary methods

- `.update()` updates the dictionary with the `key:value` pairs from other dictionary object:

```
>>> numbers.update({"four":4})
>>> numbers
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
```

- `.pop()` removes the key from the dictionary and returns its value

```
>>> numbers.pop("four")
4
>>> numbers
{'one': 1, 'two': 2, 'three': 3}
```

Dictionary methods

- `.popitem()` removes and returns a pair of `(key, value)` as a tuple from the dictionary. Pairs are returned in LIFO (last-in, first-out) order. If the dictionary is empty, then results in **`KeyError`**:

```
>>> numbers.popitem()
('three', 3)
>>> numbers
{'one': 1, 'two': 2}
```

- `del` statement delete the `key:value` pair specified by the name of the dictionary along with the key you want to delete:

```
>>> del numbers["one"]
{'two': 2}
```

- `.clear()` removes all the `key:value` pairs from the dictionary

```
>>> numbers.clear()
>>> numbers
{}
```

Dictionary methods

- The `dict_keys`, `dict_values`, and `dict_items` data types returned by various methods of dictionary are read-only, but they can be converted to lists:

```
>>> numbers = {'one': 1, 'two': 2, 'three': 3}
>>> numbers.keys()
dict_keys(['one', 'two', 'three'])
>>> list(numbers.keys())
['one', 'two', 'three']
>>> numbers.values()
dict_values([1, 2, 3])
>>> list(numbers.values())
[1, 2, 3]
>>> numbers.items()
dict_items([('one', 1), ('two', 2), ('three', 3)])
>>> list(numbers.items())
[('one', 1), ('two', 2), ('three', 3)]
```

Populating dictionaries

SUSTech Southern University of Science and Technology

- One of the common ways of populating dictionaries is to start with an empty dictionary `{}`, then use the `.update()` method to assign a value to the key using assignment operator. If the key does not exist, the `key:value` pairs will be added to the dictionary:

```
>>> countries = {}
>>> countries.update({"Asia":"China"})
>>> countries.update({"Europe":"Germany"})
>>> countries.update({"Africa":"Sudan"})
>>> countries
{'Asia': 'China', 'Europe': 'Germany', 'Africa': 'Sudan'}
```

- The dictionary can also be built using `dictionary_name[key] = value` syntax by adding `key:value` pairs to the dictionary

Exercises

The power of "for"

- A `for` loop can be used to iterate through a list, tuple, string, or a dictionary object. Here is the syntax for `for` loop:

```
    for i in L:
        #do something
```

Here, `L` can be a string, list, tuple or dictionary object. When one writes "`i in L`", where `L` is a list, `i` becomes the first element of the list and as the iteration progresses, `i` becomes the second element, the third element and so on. These elements can be then independently manipulated.

```
In [1]: l = ['a', 'b', 'c', 'd', '1', '2']
In [2]: for i in l:
   ...:     print("Now the element is " + i)
Now the element is a
Now the element is b
Now the element is c
Now the element is d
Now the element is 1
Now the element is 2
```

```
In [3]: s = 'abcd12'
In [4]: for i in s:
   ...:     print("Now the element is " + i)
Now the element is a
Now the element is b
Now the element is c
Now the element is d
Now the element is 1
Now the element is 2
```

```
In [5]: d = {'a':'A', 'b':"B", 'c':"C"}
In [6]: for k in d.keys():
   ...:     print("Now the key is {0}, value is {1}".format(k, d[k]))
   ...:
Now the key is a, value is A
Now the key is b, value is B
Now the key is c, value is C
```

Use “`for`” to create a dictionary

- **Convert two lists into a dictionary:**

 Write a Python program to convert them into a dictionary in a way that item from list1 is the key and item from list2 is the value

```
keys = ['Ten', 'Twenty', 'Thirty’]
values = [10, 20, 30]
```

- **Solution:**

```
keys = ['Ten', 'Twenty', 'Thirty’]
values = [10, 20, 30]
res_dict = dict() # empty dictionary
for i in range(len(keys)):
    res_dict.update({keys[i]: values[i]})
print(res_dict)
```

Exercise 1

- Merge two Python dictionaries into one

```
dict1 = {'Ten': 10, 'Twenty': 20, 'Thirty': 30}
dict2 = {'Thirty': 30, 'Fourty': 40, 'Fifty': 50}
```

- Expected output:

```
{'Ten': 10, 'Twenty': 20, 'Thirty': 30, 'Fourty': 40, 'Fifty': 50}
```

Exercise 1

- Merge two Python dictionaries into one

```
dict1 = {'Ten': 10, 'Twenty': 20, 'Thirty': 30}
dict2 = {'Thirty': 30, 'Fourty': 40, 'Fifty': 50}
```

- Expected output:

```
{'Ten': 10, 'Twenty': 20, 'Thirty': 30, 'Fourty': 40, 'Fifty': 50}
```

```
dict1 = {'Ten': 10, 'Twenty': 20, 'Thirty': 30}
dict2 = {'Thirty': 30, 'Fourty': 40, 'Fifty': 50}
dict3 = dict1.copy()
dict3.update(dict2)
print(dict3)
```

Exercise 2

- Delete a list of keys from a dictionary

```
sample_dict = { "name": "Kelly", "age": 25, "salary": 8000, "city": "New york"}
keys = ["name", "salary"]  # keys to remove
dict2 = {'Thirty': 30, 'Fourty': 40, 'Fifty': 50}
```

- Expected output:

```
{"age": 25, "city": "New york"}
```

Exercise 2

- Delete a list of keys from a dictionary

```
sample_dict = { "name": "Kelly", "age": 25, "salary": 8000, "city": "New york"}
keys = ["name", "salary"]  # keys to remove
dict2 = {'Thirty': 30, 'Fourty': 40, 'Fifty': 50}
```

- Expected output:

```
{"age": 25, "city": "New york"}
```

```
sample_dict = { "name": "Kelly", "age": 25, "salary": 8000, "city": "New york" }
# Keys to remove
keys = ["name", "salary"]
for k in keys:
    sample_dict.pop(k)
print(sample_dict)
```

Exercise 3

- Write a program to print the following student's name, ID, and average score:

```
student_details = {"name": "John", "ID": "100101", "marks": {"python":
95, "java": 90, "C": 85}}
```

Exercise 3

- Write a program to print the following student's name, ID, and average score:

```
student_details = {"name": "John", "ID": "100101", "marks": {"python": 95, "java": 90, "C": 85}}
```

```
student = {"name": "John", "ID": "100101", "marks": {"python": 95, "java": 90, "C": 85}}
print(f"The student's name is {student['name']}")
print(f"The student's ID is {student['ID']}")
scores = student["marks"]
average_score = sum(scores.values()) / len(scores)
print(f"The student's avarage score is {average_score}")
```

Exercise 4

- Create a dictionary containing two pairs of `key:value` by feeding the following `key` and `value` information from keyboard:

Keys (name)	Values (height)
`'Alberts'`	`185`
`'Marie'`	`170`

Exercise 4

- Create a dictionary containing two pairs of `key:value` by feeding the following `key` and `value` information from keyboard:

Keys (name)	Values (height)
'Alberts'	185
'Marie'	170

```
no_person = 2
person_info = {}
for i in range(no_person):
    dic_key = input("Please enter the name: ")
    dic_val = input("Please enter the height: ")
    dic_val = int(dic_val)
    person_info.update({dic_key: dic_val})
print(f"The created dictionary is {person_info}")
```

Exercise 5

- Check if a value exists in a dictionary:

  given a dictionary, check if value 200 exists in this dictionary:

```
sample_dict = {'a': 100, 'b': 200, 'c': 300}
```

Exercise 5

- Check if a value exists in a dictionary:

  given a dictionary, check if value 200 exists in this dictionary:

  `sample_dict = {'a': 100, 'b': 200, 'c': 300}`

```
sample_dict = {'a': 100, 'b': 200, 'c': 300}
if 200 in sample_dict.values():
    print('200 present in a dict')
```

Exercise 6

- Write a program to generate a dictionary that contains `(i:i`^{2}`)` such that `i` is a number ranging from 1 to *n* (*n* is obtained from keyboard input):

Exercise 6

- Write a program to generate a dictionary that contains (i:i^2) such that `i` is a number ranging from 1 to *n* (*n* is obtained from keyboard input):

```
num = input("The largest number you want list: ")
num = int(num)
dict_square = dict()
for i in range(num):
    dict_square[i] = i**2
print(f"The generated dictionary for (i:i**2) is {dict_square}")
```

Exercise 7

- Write a program to count the number of characters in a word using dictionary. Display the character and their appeared times in alphabetical order:

```
pneumonoultramicroscopicsilicovolcanoconiosis
```

Exercise 7

- Write a program to count the number of characters in a word using dictionary. Display the character and their appeared times in alphabetical order:

  pneumonoultramicroscopicsilicovolcanoconiosis

```
word = "pneumonoultramicroscopicsilicovolcanoconiosis"
char_dict = dict()
for character in word:
    char_dict[character] = char_dict.get(character, 0) + 1
char_dict_sorted_keys = sorted(char_dict.keys())
for key in char_dict_sorted_keys:
    value = char_dict.get(key)
    print(f"The character '{key}' appears {value} times in the word")
```