PYTHON PROGRAMMING

STRING AND STRING OPERATIONS

OUTLINE

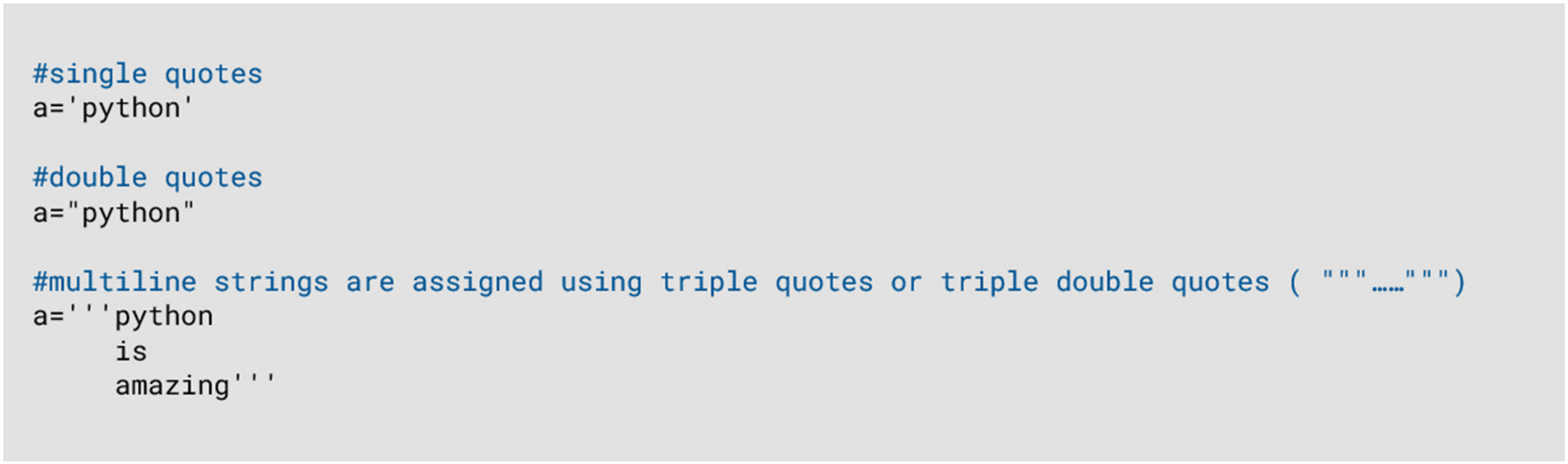
SOME STANDARD DATA TYPES IN PYTHON

- Numbers
- Boolean
- String
- List
- Tuple
- Dictionary
- Set

Strings

- Contiguous set of characters surrounded by single, double or triple quotation marks.
- Immutable (unchangeable).
- Follows zero-based indexing (for accessing single or more characters).

Assigning strings to variables (creating strings)



```
#single quotes
a='python'

#double quotes
a="python"

#multiline strings are assigned using triple quotes or triple double quotes ( """……""")
a='''python
     is
     amazing'''
```

Strings

- Fortlaufender Satz von Zeichen, umgeben von einfachen, doppelten oder dreifachen Anführungszeichen.
- Unveränderlich
- Folgt der nullbasierten Indizierung (für den Zugriff auf einzelne oder mehrere Zeichen).

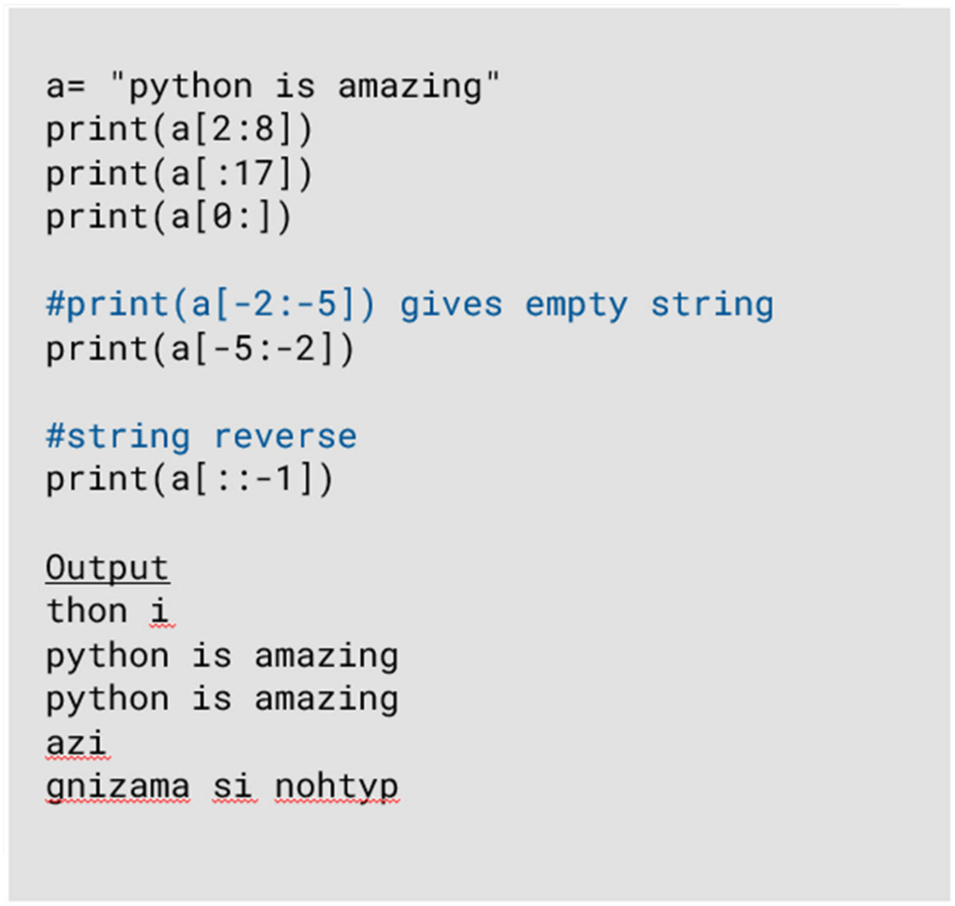
Assigning strings to variables (creating strings)

```
#single quotes
a='python'

#double quotes
a="python"

#multiline strings are assigned using triple quotes or triple double quotes ( """……""")
a='''python
     is
     amazing'''
```

Accessing String and its elements

```
a= "python is amazing"
print(a)
print(a[1])

#negative indexing starts from -1 to access characters starting from end of a string
print(a[-1])

Output
python is amazing
y
g
```

Accessing subsets of strings using slicing operation



```
a= "python is amazing"
print(a[2:8])
print(a[:17])
print(a[0:])

#print(a[-2:-5]) gives empty string
print(a[-5:-2])

#string reverse
print(a[::-1])

Output
thon i
python is amazing
python is amazing
azi
gnizama si nohtyp
```

Escape Sequences

Escape sequence is a combination of characters that has different meaning than the literal characters in them. It allows including special characters which has a predefined meaning into a string. Characters are escaped using backslash(\).

\n	– linefeed
\t	– tab
\\	– backslash
\'	– displays single quotes in the output screen
\"	– displays double quotes in the output screen

```
print("Python \n Programming")
print("Language \t Python")

Output
Python
Programming
Language         Python
```

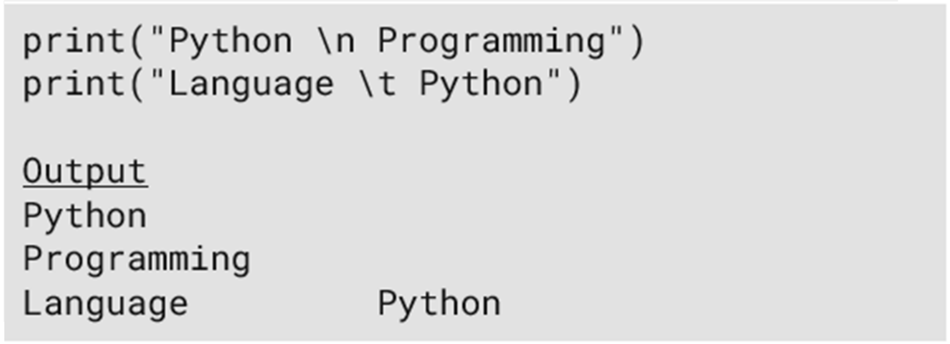
Alternatively, to display single quote use double quotes as delimiter and vice versa.

```
print("She said 'hello' and left")
print('She said "hello" and left')

Output
She said 'hello' and left
She said "hello" and left
```

Escape Sequences

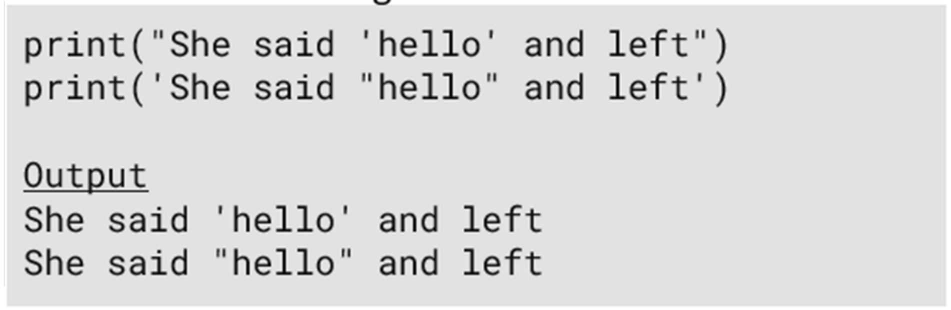
Escape-Sequenz ist eine Kombination von Zeichen, die eine andere Bedeutung hat als die darin enthaltenen Buchstaben. Es ermöglicht das Einfügen von Sonderzeichen, die eine vordefinierte Bedeutung haben, in eine Zeichenfolge. Zeichen werden mit Backslash (\) maskiert.

\n – linefeed
\t – tab
\\ – backslash
\' – displays single quotes in the output screen
\" – displays double quotes in the output screen



```
print("Python \n Programming")
print("Language \t Python")

Output
Python
Programming
Language          Python
```

Um einfache Anführungszeichen anzuzeigen, verwenden Sie alternativ doppelte Anführungszeichen als Trennzeichen und umgekehrt.



```
print("She said 'hello' and left")
print('She said "hello" and left')

Output
She said 'hello' and left
She said "hello" and left
```

Deleting a string

```
del(a)
```

Some operators used with strings

```
a=“py"
b="program"

#concatenation
print(a+b)

#repetition
print(a*2)

#membership operators
print('y' in a)
print('t' in a )
print('t' not in a)
```

```
Output

pyprogram
pypy
True
False
True
```

Functions Discussed

1) len(string)
2) capitalize()
3) title()
4) count(value, start, end)
5) find(value, start, end)
6) index(value, start, end)
7) lower()
8) islower()
9) startswith(value, start, end)
10) strip(value)
11) replace(oldval, newval, count)
12) split(separator, maxsplit)
13) format(value1, value2,…)

Some string functions

```python
a="python is amazing"
b="PYTHON programming"
c="python;java;c;are programming languages"

print(len(a))

#capitalizes first character
print(a.capitalize())

#capitalizes first character of each word
print(a.title())

'''2nd and 3rd arguments are optional. Default values are zero and
end of string'''
print(a.count("is",5,16))

'''2nd and 3rd arguments are optional. Default values are zero and
end of string'''
print(a.find("is",5,16))

'''2nd and 3rd arguments are optional. Default values are zero and
end of string'''
print(a.index("is",5,16))
```

Output

```
17
Python is amazing
Python Is Amazing
1
7
7
```

```
converts to lower case
print(b.casefold())
print(b.lower())

print(b.islower())

'''checks whether substring from index 2 to 5 starts
with T.2nd and 3rd arguments optional'''
print(b.startswith("T",2,5))

#split string into 3 based on given substring
print(a.partition("is"))

'''removes leading & trailing characters of given
type. Default is space.'''
d="**@python@**"
print(d.strip("*"))

'''3rd argument optional. Represents how many
occurrences must be replaced'''
print(a.replace("amazing", "fantastic", 1))
#
'''optional arguments, defaults to space(separator)
and -1(all occurance of given separator)'''
print(c.split(';',1))
```

```
python programming
python programming
False
True
('python ', 'is', ' amazing')
@python@
python is fantastic
['python', 'java;c;are programming languages']
```

Thank You

PYTHON PROGRAMMING

Control Flow

- If-statement
- For-loop
- While-loop
- beak
- continue

OUTLINE

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

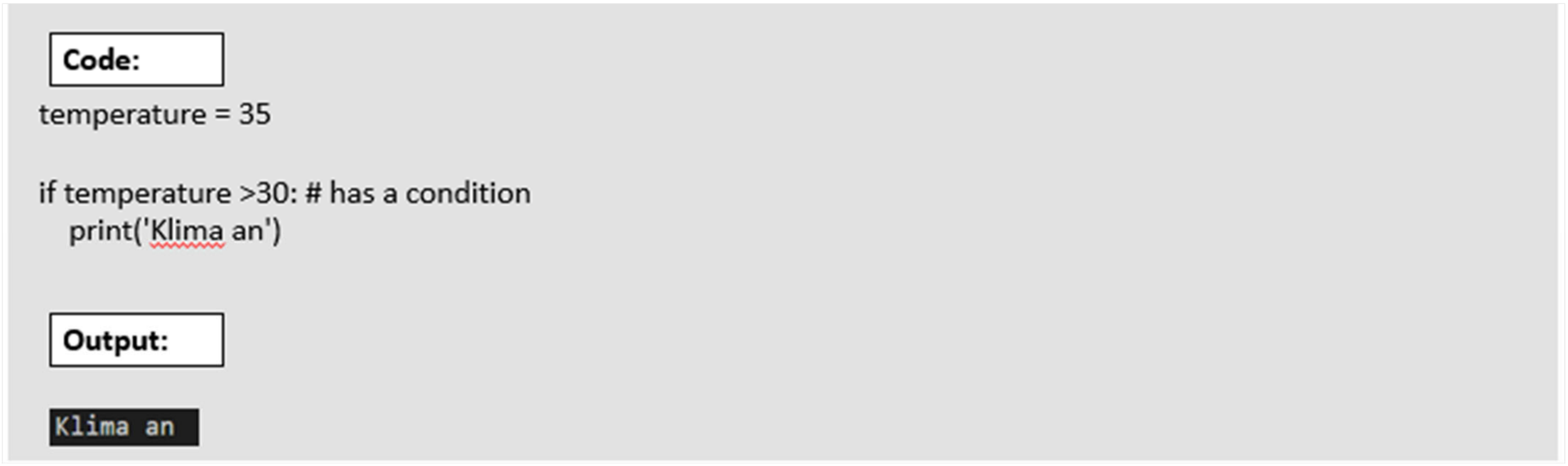
- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.
An "if statement" is written by using the if keyword.

If Statement

- In this example we use the variable temperature, which is used as part of the if statement to test whether the temperature is greater than 30. As temperature is 35, and the check temperature in the if statement is 30, we know that 35 is greater than 30, and so we print to screen that “Klima an".

Example 1:



Code:

```
temperature = 35

if temperature >30: # has a condition
   print('Klima an')
```

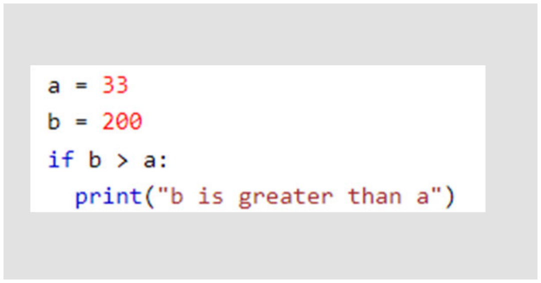
Output:

```
Klima an
```

If Statement

- In this example we use two variables, a and b, which are used as part of the if statement to test whether b is greater than a. As a is 33, and b is 200, we know that 200 is greater than 33, and so we print to screen that "b is greater than a".

Example 1:

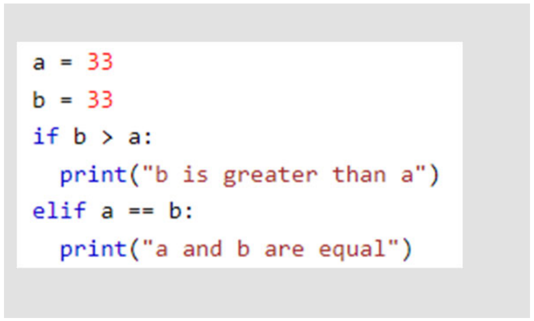


```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

Elif Statement

- The elif keyword is Python's way of saying "if the previous conditions were not true, then try this condition".
- In this example a is equal to b, so the first condition is not true, but the elif condition is true, so we print to screen that "a and b are equal".

Exampel 2:

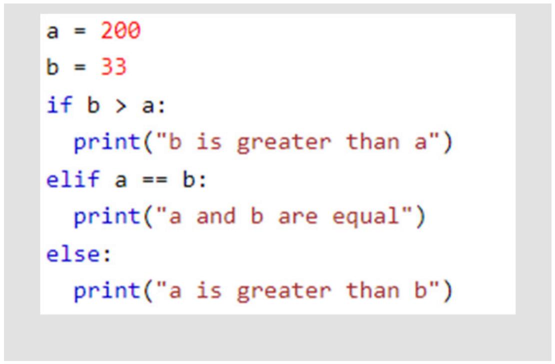


```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

Else

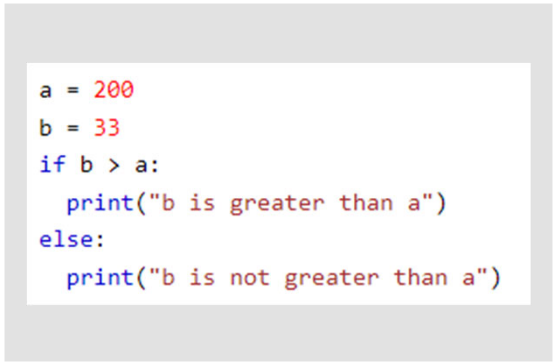
- The else keyword catches anything which isn't caught by the preceding conditions.
- In this example a is greater than b, so the first condition is not true, also the elif condition is not true, so we go to the else condition and print to screen that "a is greater than b".
- You can also have an else without the elif:

Example 3



```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Example 4

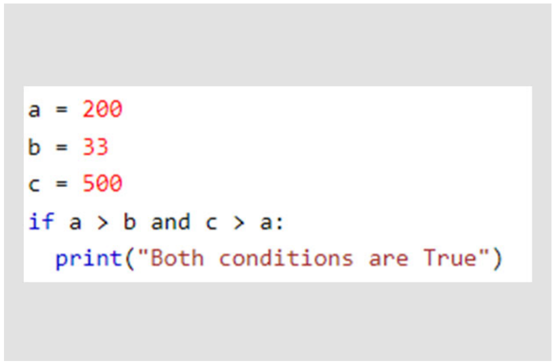


```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

And

- The and keyword is a logical operator, and is used to combine conditional statements:
- In this example we test if a is greater than b, AND if c is greater than a:

Example 5

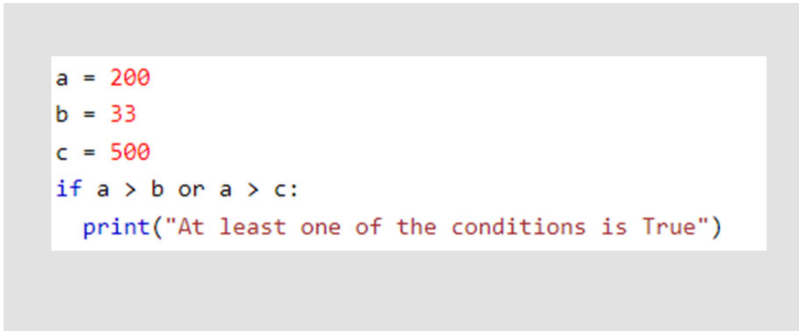


```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

OR

- The or keyword is a logical operator, and is used to combine conditional statements:
- In this example we test if a is greater than b, OR if a is greater than c:

Example 6



```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

Python While Loops

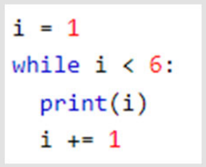
Python has two primitive loop commands:

- while loops
- for loops

The While Loop

- With the while loop we can execute a set of statements as long as a condition is true.
- In this example we print i as long as i is less than 6
- The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

Example 7



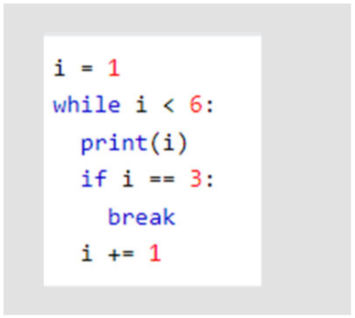
```
i = 1
while i < 6:
  print(i)
  i += 1
```

Note: remember to increment i, or else the loop will continue forever.

The break Statement

- With the break statement we can stop the loop even if the while condition is true:
- In this example we exit the loop when i is 3:

Example 8

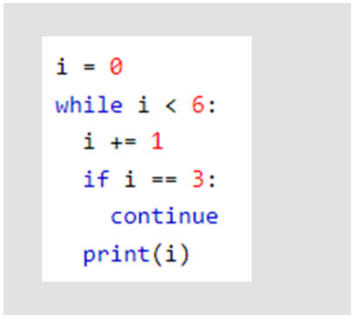


```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

The continue Statement

- With the continue statement we can stop the current iteration, and continue with the next:
- In this example we continue to the next iteration if i is 3:

Example 9



```
i = 0
while i < 6:
  i += 1
  if i == 3:
    continue
  print(i)
```

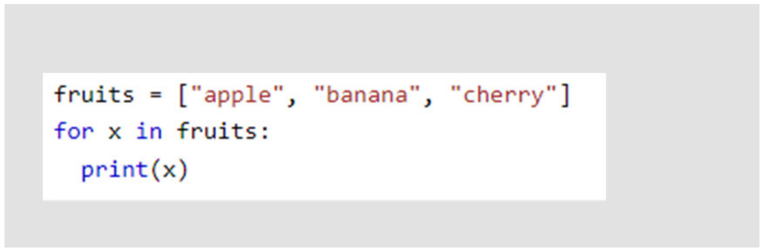
The Python For Loops

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

- This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.
- The for loop does not require an indexing variable to set beforehand.
- In this example we print each fruit in a fruit list:

Example 10

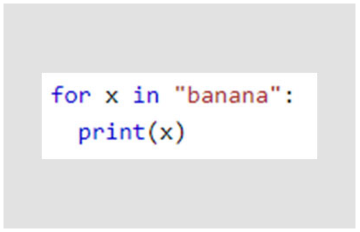


```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Looping Through a String

- Even strings are iterable objects, they contain a sequence of characters:
- In this example we Loop through the letters in the word "banana":

Example 11

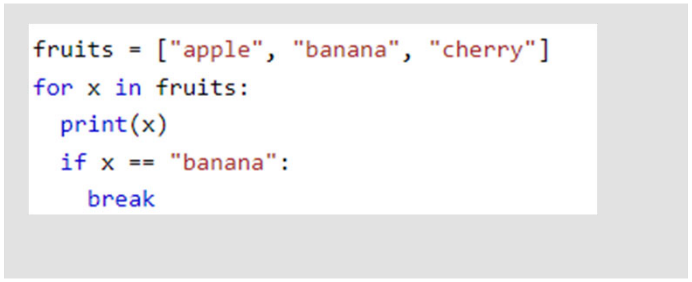


```
for x in "banana":
  print(x)
```

The break Statement

- With the break statement we can stop the loop before it has looped through all the items:
- In this example we exit the loop when x is "banana":

Example 12

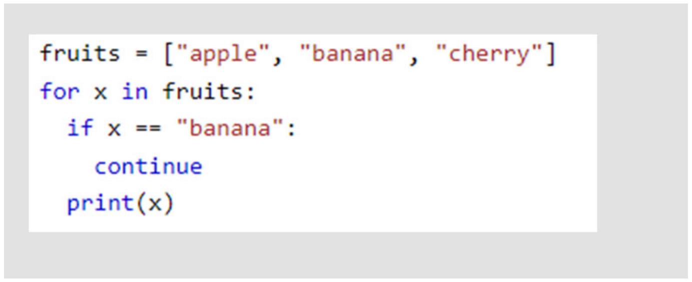


```python
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

The continue Statement

- With the continue statement we can stop the current iteration of the loop, and continue with the next:
- In this example we do not print banana:

Example 13

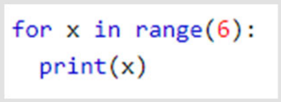


```python
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
    continue
  print(x)
```

The range() Function

- To loop through a set of code a specified number of times, we can use the range() function,
- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.
- In this example we loop using the range() function:

Example 14



```python
for x in range(6):
  print(x)
```

Note: that range(6) is not the values of 0 to 6, but the values 0 to 5.

PYTHON PROGRAMMING

LIST AND OPERATIONS ON LIST

OUTLINE

List

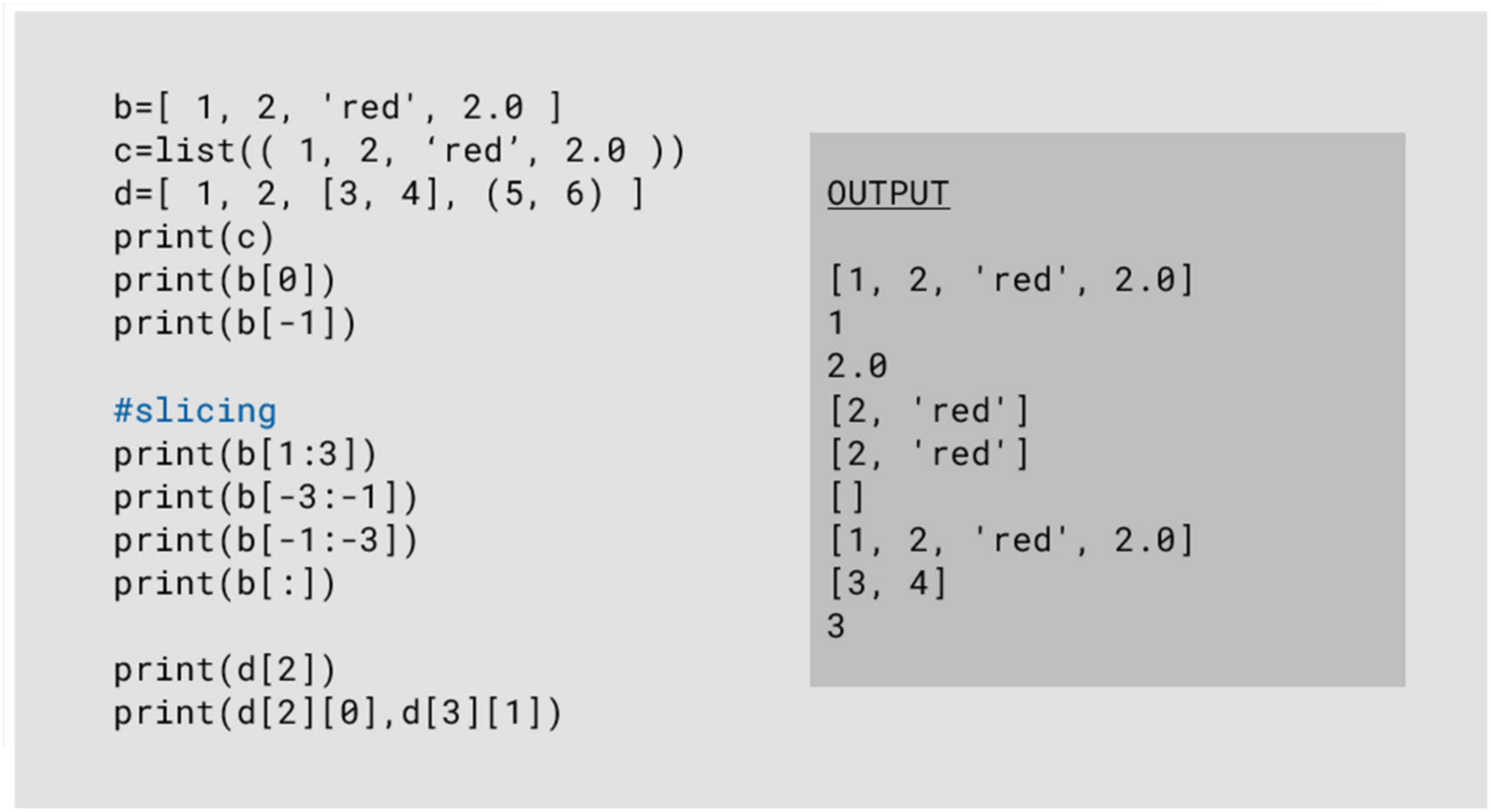
- Ordered list of elements indexed from 0 to length-1.
- Repeated elements are allowed.
- Mutable (can be modified or changed)
- Can contain elements of different data types such as integer, float, string, list, tuple etc.

Creating List

Elements are written separated by commas and enclosed within square brackets

```
#empty list
a=[]

#different types
b=[1, 2, 'red', 2.0]

#creating list using list constructor. It takes only one argument, hence pass list values
in () or [ ].
c=list((1, 2, 'red', 2.0))

#can contain another list, tuple or even functions
d=[1, 2, [3, 4], (5, 6)]
```

Accessing List and its elements (slicing)



```
b=[ 1, 2, 'red', 2.0 ]
c=list(( 1, 2, ‘red’, 2.0 ))
d=[ 1, 2, [3, 4], (5, 6) ]
print(c)
print(b[0])
print(b[-1])

#slicing
print(b[1:3])
print(b[-3:-1])
print(b[-1:-3])
print(b[:])

print(d[2])
print(d[2][0],d[3][1])
```

OUTPUT

```
[1, 2, 'red', 2.0]
1
2.0
[2, 'red']
[2, 'red']
[]
[1, 2, 'red', 2.0]
[3, 4]
3
```

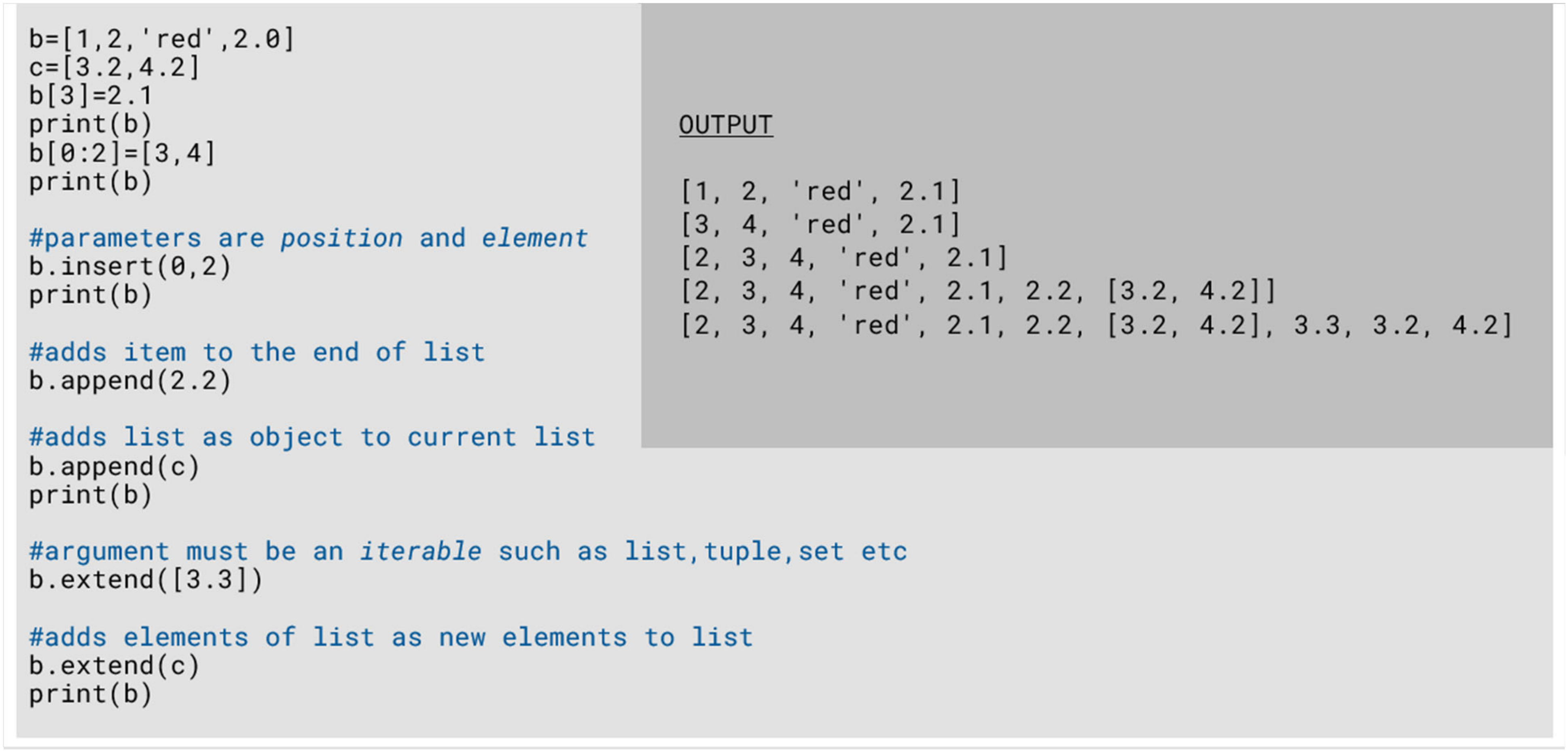
Deleting list and its elements (pop(), del, remove())

```
b=[1,2,’red’,2.0]

#pop can store deleted item, default parameter is -1,last item
item=b.pop(0)
print(item)
print(b)

#del cannot store deleted item hence cannot write in print function
del b[0]
print(b)

#cannot store and delete item by specifying value, not index
b.remove(‘red’)
print(b)

#deletes entire list
del b
```

OUTPUT

```
1
[2, 'red', 2.0]
['red', 2.0]
[2.0]
```

Adding and Updating List: (append(), insert(), extend())



```python
b=[1,2,'red',2.0]
c=[3.2,4.2]
b[3]=2.1
print(b)
b[0:2]=[3,4]
print(b)

#parameters are position and element
b.insert(0,2)
print(b)

#adds item to the end of list
b.append(2.2)

#adds list as object to current list
b.append(c)
print(b)

#argument must be an iterable such as list,tuple,set etc
b.extend([3.3])

#adds elements of list as new elements to list
b.extend(c)
print(b)
```

```
OUTPUT

[1, 2, 'red', 2.1]
[3, 4, 'red', 2.1]
[2, 3, 4, 'red', 2.1]
[2, 3, 4, 'red', 2.1, 2.2, [3.2, 4.2]]
[2, 3, 4, 'red', 2.1, 2.2, [3.2, 4.2], 3.3, 3.2, 4.2]
```

Difference between append() and extend()

Both append() and extend() adds elements to the end of the list

append() can accept both iterable object and individual element as arguments.
extend() can only accept an iterable object as argument.

append() adds an iterable object as the same object to the end of the list.
extend() separates individual elements of the iterable object and adds as different elements to the end of the list.

In the previous code, list c = [3.2, 4.2] is added as a list itself into the current list b using append() method.
Also, extend() method separated elements 3.2, 4.2 and added as float type individual elements to b.

Copying list

Case 1

```
list1=[1,2,3]
list2=list1
print(list1)
print(list2)
list1.append(4)
print(list1)

'''both list1 & list2 points to
same list'''
print(list2)
```

OUTPUT

```
[1, 2, 3]
[1, 2, 3]
[1, 2, 3, 4]
[1, 2, 3, 4]
```

Case 2

```
list1=[1,2,3]
list2=list1[:]
print(list1)
print(list2)
list1.append(4)
print(list1)
print(list2)
```

OUTPUT

```
[1, 2, 3]
[1, 2, 3]
[1, 2, 3, 4]
[1, 2, 3]
```

Case 3

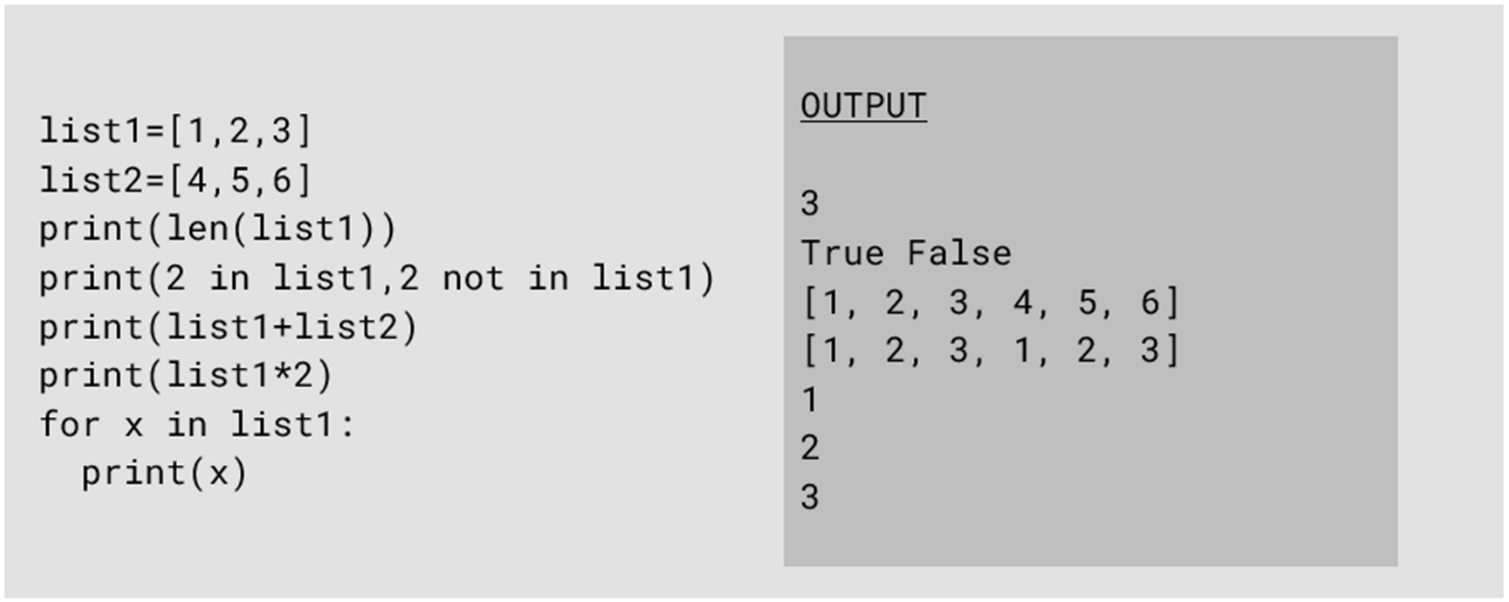
```
from copy import copy
list1=[1,2,3]
list2=copy(list1)
print(list1)
print(list2)
list1.append(4)
print(list1)
print(list2)
```

OUTPUT

```
[1, 2, 3]
[1, 2, 3]
[1, 2, 3, 4]
[1, 2, 3]
```

Basic Operations

len(), concatenation(+), repetition(*), membership(in, not in),iteration



```
list1=[1,2,3]
list2=[4,5,6]
print(len(list1))
print(2 in list1,2 not in list1)
print(list1+list2)
print(list1*2)
for x in list1:
  print(x)
```

OUTPUT

```
3
True False
[1, 2, 3, 4, 5, 6]
[1, 2, 3, 1, 2, 3]
1
2
3
```

Functions Discussed Earlier

1) pop(position)
2) remove(element)
3) append(element)
4) insert(position, element)
5) extend(iterable)
6) copy( )
7) len(list)

Functions Discussed Next

1) index(element)
2) max(list)
3) min(list)
4) list(sequence)
5) clear( )
6) count(element)
7) reverse( )
8) sort(reverse=True/False, key=myFunc)
9) sorted(list, reverse=True/False, key=myFunc)

List built-in functions and methods

```
list1=[1,2,3,3]
print(list1.index(3))
print(max(list1),min(list1))
string1="abcdef"
a=list(string1)
print(a)
a.clear()
print(a)
print(list1.count(3))
list1.reverse()
print(list1)
```

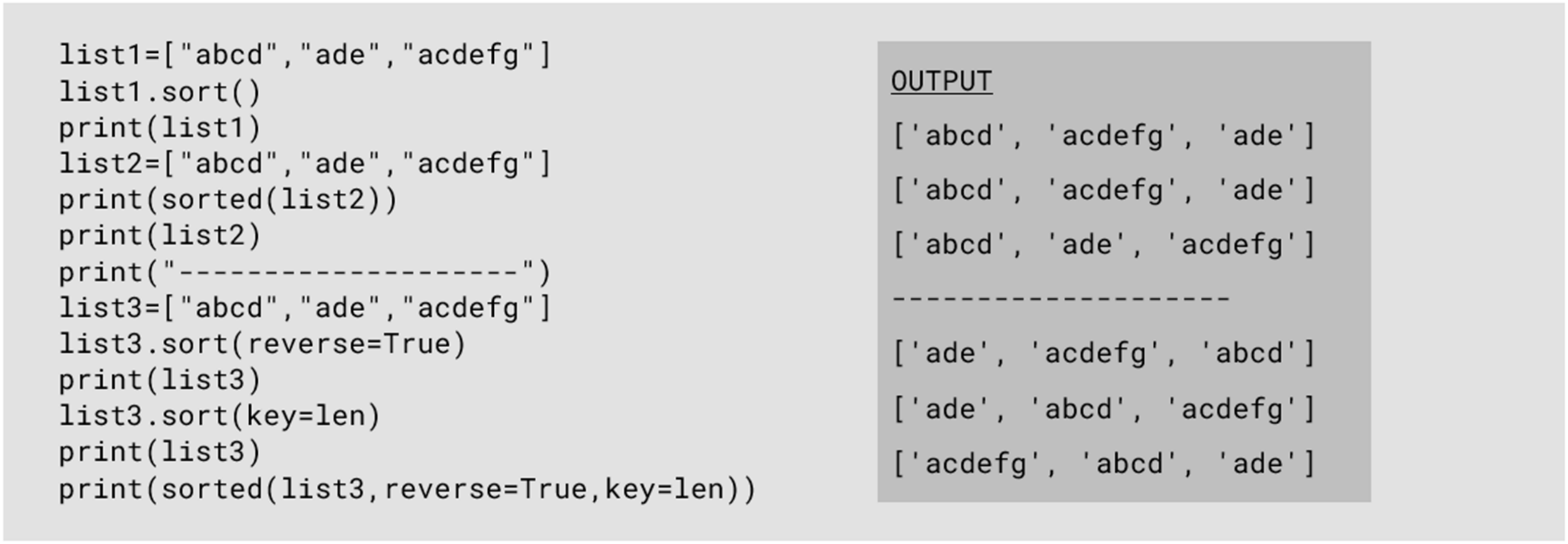
OUTPUT

```
3 1
['a', 'b', 'c', 'd', 'e', 'f']
[]
2
[3, 3, 2, 1]
```

- sort(reverse=true/false, key=myFunc)
- sorted(list, reverse=true/false, key=myFunc)

For both built in functions :
reverse is optional and if ***true***, sorted in descending order
key is optional, if given, acts as a function based on which the sorting comparison occurs

Difference :
sort() changes original list
sorted(list) returns sorted list rather than changing the original



```
list1=["abcd","ade","acdefg"]
list1.sort()
print(list1)
list2=["abcd","ade","acdefg"]
print(sorted(list2))
print(list2)
print("--------------------")
list3=["abcd","ade","acdefg"]
list3.sort(reverse=True)
print(list3)
list3.sort(key=len)
print(list3)
print(sorted(list3,reverse=True,key=len))
```

```
OUTPUT
['abcd', 'acdefg', 'ade']
['abcd', 'acdefg', 'ade']
['abcd', 'ade', 'acdefg']
--------------------
['ade', 'acdefg', 'abcd']
['ade', 'abcd', 'acdefg']
['acdefg', 'abcd', 'ade']
```

Selection sort

```python
list1=[30,20,15,10,50]
print("original list : ",list1)
n=len(list1)
for i in range(n-1):
    small=i
    for j in range(i+1,n):
        if list1[j]<list1[small]:
            small=j
    if i!=small:
        temp=list1[i]
        list1[i]=list1[small]
        list1[small]=temp
print("sorted list : ",list1)
```

OUTPUT

```
original list :  [30, 20, 15, 10, 50]
sorted list :  [10, 15, 20, 30, 50]
```

Thank You

PYTHON PROGRAMMING

TUPLES AND OPERATIONS ON TUPLES

OUTLINE

Tuples

- Indexed
- Immutable (cannot update, add or delete elements)
- Can have heterogeneous elements (different data types)
- Defined by enclosing elements in parentheses () or simply by separating elements with comma only
- Iterating over elements is faster in a tuple than in a list

Creating Tuple

```
° &HKOT!OPKG@£
OPK˙,„

°4DIBG@!@G@H@IO!£
°RDOCJPO!<!>JHH<!OPK!RDGG!=@!>JIND?@M@?!<N!!DIO@B@M!OTK@
OPK˙,WW¡„!!!

°.PGODKG@!@G@H@ION£
OPK˙,W¡X¡,<<<¡<=<„¡”Y¡Z»„
JM
OPK˙W¡X¡,<<<¡<=<„¡”Y¡Z»!°RDOCJPO!K<M@IOC@NDN
```

Tuple Packing and Unpacking

When several tuple values are assigned to a single variable it is known as **packing**

```
OPK˙,W¡X¡,<<<¡<=<„¡”Y¡Z»„
```

When packed variable or a tuple is assigned to another tuple of same number of variables, it is known as **unpacking**

```
,<¡=¡>¡?„˙OPK
KMDIO,<„
KMDIO,=„
KMDIO,>„

°@<>C!Q<MD<=G@!DI!OPKG@!><I!=@!<>>@NN@?!DI?DQD?P<GGT
KMDIO,?„
```

```
065165
W
X
,<<<¡!<=<„
”Y¡!Z»
```

Swapping made easy with tuple unpacking:

```
IJMH<G!NR<KKDIB!£!O@HK˙S
                  S˙T
                  T˙O@HK
```

```
PNDIB!PIK<>FDIB!£S¡T!˙!T¡S
```

Accessing tuple and its elements

```
OPK˙,W¡X¡,<<<¡<=<„¡”Y¡Z»„
KMDIO,OPK„
KMDIO,OPK”V»„
KMDIO,OPK”X£Z»„
KMDIO,OPK”'Z£'X»„
KMDIO,OPK”£»„
```

```
065165

,W¡!X¡!,<<<¡!<=<„¡!”Y¡!Z»„
W
,,<<<¡!<=<„¡!”Y¡!Z»„
,W¡!X„
,W¡!X¡!,<<<¡!<=<„¡!”Y¡!Z»„
```

Basic Operations

len(), concatenation(+), repetition(*), membership(in, not in), iteration

```
N˙,W¡X¡Y„
O˙,W¡X¡Y„
KMDIO,G@I,N„„
KMDIO,N´O„
KMDIO,N-X„
AJM!D!DI!N£
    KMDIO,D„
KMDIO,W!DI!N„
KMDIO,W!IJO!DI!N„
```

```
065165
Y
,W¡!X¡!Y¡!W¡!X¡!Y„
,W¡!X¡!Y¡!W¡!X¡!Y„
W
X
Y
5MP@
’<GN@
```

Delete and update (mutable elements only)

```
OPK˙,W¡X¡Y„

<<<NCJRN!@MMJM¡!
OPKG@!DN!DHHPO<=G@<<<
?@G!OPK”W»

°?@G@O@N!RCJG@!OPKG@
?@G!OPK
```

```
OPK˙,W¡X¡,<<<¡<=<„¡”Y¡Z»

<<<?@G@O@N!NDI>@!OCDN!
@G@H@IO!DN!<!GDNO RCD>C!
DN!HPO<=G@<<<
?@G!OPK”Y»”V»
KMDIO,OPK„


065165
,W¡!X¡!,<<<¡!<=<„¡!”Z»„
```

```
OPK˙,W¡X¡,<<<¡<=<„¡”Y¡Z»„

<<<NCJRN!@MMJM¡!OPKG@!DN!
DHHPO<=G@<<<
OPK”W»˙Y
```

```
OPK˙,W¡X¡,<<<¡<=<„¡”Y¡Z»„

<<<PK?<O@N¡!OCDN!OPKG@!
@G@H@IO!DN!<!GDNO RCD>C!
DN!HPO<=G@<<<
OPK”Y»”V»˙Z
KMDIO,OPK„

065165
,W¡!X¡!,<<<¡!<=<„¡!”Z¡!Z»„
```

Reverse of a tuple (using slicing and reversed())

```
OPK˙,_^¡_¡^\¡]\ZZ¡]„
KMDIO,OPK„

°NGD>DIB
KMDIO,OPK”££'W»„!!

KMDIO,OPK„

°M@Q@MN@?,„
KMDIO,OPKG@,M@Q@MN@?,OPK„„„!!!
```

```
065165

,_^¡!_¡!^\¡!]\ZZ¡!]„
,]¡!]\ZZ¡!^\¡!_¡!_^„
,_^¡!_¡!^\¡!]\ZZ¡!]„
,]¡!]\ZZ¡!^\¡!_¡!_^„
```

Functions Discussed

1) zip(iterable1, iterable2,…)
2) max(_tuple_)
3) min(_tuple_)
4) tuple(sequence)
5) index(element)
6) count(element)

Tuple built-in functions and methods

```
N˙,W¡X¡Y„
O˙,W¡X¡Y„
<˙>C@GGJ>

<<<O<F@N!V!JM!HJM@!DO@M<=G@N!<I?!
>JH=DI@N!>JMM@NKJI?DIB!@G@H@ION!DIOJ!OPKG@N¢
.PNO!=@!K<NN@?!OJ!OPKG@¡GDNO!JM!N@O!OJ!?DNKG<T!M@NPGODIB!OPKG@N¢<<<
KMDIO,OPKG@,UDK,N¡O„„„
KMDIO,H<S,N„¡HDI,N„„

°>JIQ@MON!BDQ@I!N@LP@I>@!OJ!OPKG@!
KMDIO,OPKG@,<„„
OPK˙,,W¡X¡X¡X¡!,<<<¡!<=<„¡!”Y¡!Z»„„
KMDIO,OPK¢DI?@S,X„„
KMDIO,OPK¢>JPIO,X„„
```

```
065165

,,W¡!W„¡!,X¡!X„¡!,Y¡!Y„„
Y!W
,<C<¡!<@<¡!<G<¡!<G<¡!<J<„
W
Y
```

Thank You

PYTHON PROGRAMMING

DICTIONARY AND ASSOCIATED OPERATIONS

OUTLINE

- Dictionary
- Create and Assign
- Access Elements
- Change/Add Elements
- Delete/Remove Elements
- Basic Operations
- Built-in Functions and Methods

Dictionary

- Mutable
- Unordered Key-Value pairs
- Indexed according to keys

Keys and Values

A dictionary is a set of key-value pairs enclosed in curly braces {}, where each key is separated from value using a colon (:) and each pair is separated from each other by comma (,)

Keys must be unique, immutable type (integer, string, tuple etc.) and are case sensitive.

Values may not be unique and hence, can be repeated.

Creating Dictionaries

```
°@HKOT!?D>ODJI<MT!
?˙…‰

°><I!C<Q@!?DAA@M@IO!OTK@N!
?W˙…W£X¡Y£Z¡<<<£[¡<=<£\‰

°F@TN!<M@!DHHPO<=G@¡!GDNO!><IIJO!=@!
PN@?!<N!<!F@T!
?X˙…,<<<¡<=<„£”W¡X»!‰

°I@NO@?!?D>ODJI<MT!
?Y˙…W£…X£Y¡Z£[‰¡X£\‰
KMDIO,?W¡>ªI>¡?X¡>ªI>¡?Y„

065165
…W£!X¡!Y£!Z¡!<<<£![¡!<=<£!\‰!
…,<<<¡!<=<„£!”W¡!X»‰
…W£!…X£!Y¡!Z£![‰¡!X£!\‰
```

```
$M@<ODIB!?D>ODJI<MD@N!PNDIB!?D>O,„!API>ODJI

?Z˙?D>O,…W£X¡Y£Z‰„

<<<?D>O,„!O<F@N!JIGT!JI@!<MBPH@IO¡!C@I>@!
K<NN!F@T'Q<G!K<DMN!DIND?@!GDNO!JM!OPKG@<<<
?[˙?D>O,”,W¡X„¡,Y¡Z„»„!

KMDIO,?Z¡>ªI>¡?[„

065165
…W£!X¡!Y£!Z‰!
…W£!X¡!Y£!Z‰
```

Accessing dictionary elements

Keys are used as an index to access values. Slicing operation is not supported in dictionary.

```
?W'…W£X¡Y£Z¡<<<£[¡<=<£\‰
?X'…,W¡X„£<IPH=@MN<¡,<<<¡<=<„£<<GKC<=@ON<‰!!!
?Y'…W£…X£Y¡Z£[‰¡X£\‰
KMDIO,?W”<<<»¡>ªI>¡?X”,W¡X„»„
KMDIO,?Y”W»¡>ªI>¡?Y”X»¡>ªI>¡?Y”W»”X»„

°PNDIB!B@O,„!API>ODJI!
KMDIO,?W¢B@O,W„¡>ªI>¡?W¢B@O,<<<„„
```

```
065165
[!
IPH=@MN
…X£!Y¡!Z£![‰!
\!
Y
X!
[
```

Change or add elements

If the key is already present its value gets updated otherwise a new key-value pair is added.

```
?W'…W£X¡Y£Z¡<<<£[¡<=<£\‰
?W”W»'WV
?W”X»'XV
KMDIO,?W„
```

```
065165
…W£!WV¡!Y£!Z¡!<<<£![¡!<=<£!\¡!X£!XV‰
```

Delete or Remove elements

```
?W'…W£X¡Y£Z¡<<<£[¡<=<£\‰
?X'…W£X¡Y£Z¡[£\‰

°?@G@O@N!@G@H@IO!RDOC!BDQ@I!F@T!ó!NOJM@N!OC@!Q<GP@!
KMDIO,?W¢KJK,W„„

°?@G@O@N!<I!<M=DOM<MT!DO@H!ó!NOJM@N!OC@!Q<GP@!
KMDIO,?W¢KJKDO@H,„„

°?@G@O@N!@G@H@IO!RDOC!BDQ@I!F@T!ó!><IIJO!NOJM@!Q<GP@!
?@G!?W”Y»
KMDIO,>%D>ODJI<MT!<AO@M!?@G@ODIB!Y£Z!DN!'''  >¡?W„

°?@G@O@N!@IODM@!?D>ODJI<MT!
?@G!?W

°?@G@O@N!@IODM@!?D>ODJI<MT!
?X¢>G@<M,„
```

```
065165
X
,<=<¡!\„
%D>ODJI<MT!<AO@M!?@G@ODIB!Y£Z!DN!'''  …<<<£![‰
```

Basic operations (+ and * cannot be applied)

```
?W˙…W£X¡Y£Z¡<<<£[¡<=<£\‰

°M@OPMIN!G@IBOC!
KMDIO,G@I,?W„„

°H@H=@MNCDK!JK@M<OJMN!<M@!AJM!F@TN!JIGT!
KMDIO,W!DI!?W¡>ªI>¡X!DI!?W„

AJM!D!DI!?W£
    °!D!KMDION!F@T!<I?!?W”D»!KMDION!Q<GP@
    KMDIO,D¡>£>¡?W”D»„
```

```
065165
Z
5MP@!
’<GN@
W!£!X
Y!£!Z
<!£![
=!£!\
```

Functions Discussed Earlier

1) dict(key_value_pairs)
2) get(key)
3) pop(key)
4) popitem( )
5) clear( )
6) len(dictionary)

Functions Discussed Next

1) any(dictionary)
2) all(dictionary)
3) sorted(list, reverse=True/False, key=myFunc)
4) keys()
5) values()
6) items()
7) copy()
8) fromkeys(dictionary, value_for_all_keys)
9) update(key_value_pair)
10) setdefault(key, value)

Built in functions and methods

```
?˙…<<£<<¡X£Y‰

°5MP@!DA!@Q@I!JI@!F@T!DI!?D>ODJI<MT!C<N!#JJG@<I!Q<GP@!JA!5MP@
KMDIO,<IT,?„„!

°5MP@!JIGT!DA!<GG!F@TN!DI!?D>ODJI<MT!C<Q@!#JJG@<I!Q<GP@!JA!5MP@
KMDIO,<GG,?„„!

?W˙…X£W¡Y£W¡W£W¡Z£W‰
°NJMON!F@TN!DI!<N>@I?DIB!JM?@M!ó!M@OPMIN!<N!GDNO!
KMDIO,NJMO@?,?W„„
°NJMO@?,„!?J@NIflO!HJ?DAT!JMDBDI<G!?D>ODJI<MT!
KMDIO,?W„
```

```
065165
5MP@
’<GN@
”W¡!X¡!Y¡!Z»
…X£!W¡!Y£!W¡!W£!W¡!Z£!W‰
```

```
?W˙…W£X¡Y£Z¡<<<£[¡<=<£\‰

°M@OPMIN!F@TN!<N!GDNO!
KMDIO,?W¢F@TN,„„

°M@OPMIN!Q<GP@N!<N!GDNO!
KMDIO,?W¢Q<GP@N,„„

°M@OPMIN!OPKG@!JA!F@T'Q<GP@!K<DMN
KMDIO,?W¢DO@HN,„„!

°>JKD@N!?W!OJ!?
?˙?W¢>JKT,„!
KMDIO,?„
?W”W»˙Y
<<<PK?<O@N!IJO!M@AG@>O@?!DI!
>JKD@?!?D>ODJI<MT<<<
KMDIO,?W¡?„
```

```
065165

?D>O¤F@TN,”W¡!Y¡!<<<¡!<=<»„

?D>O¤Q<GP@N,”X¡!Z¡![¡!\»„

?D>O¤DO@HN,”,W¡!X„¡!,Y¡!Z„¡!,<<<¡![„¡!,<=<¡!\„»„

…W£!X¡!Y£!Z¡!<<<£![¡!<=<£!\‰

…W£!Y¡!Y£!Z¡!<<<£![¡!<=<£!\‰!…W£!X¡!Y£!Z¡!<<<£![¡!<=<£!\‰
```

Continued...

```
<<<>M@<O@N!I@R!?D>ODJI<MT!AMJH!<IJOC@M!?D>ODJI<MT¡
XI? <MBPH@IO!DN!>JMM@NKJI?DIB!Q<GP@N!AJM!F@TN¡!?@A<PGO!DN!/JI@ <<<
I@R?D>O˙?W¢AMJHF@TN,?W¡V„!
KMDIO,I@R?D>O„
I@R?D>O˙?W¢AMJHF@TN,?W„
KMDIO,I@R?D>O„

°PK?<O@N!?D>ODJI<MT!RDOC!BDQ@I!K<DM!
?W¢PK?<O@,…<><£]‰„
KMDIO,?W„

°M@OPMIN!JMDBDI<G!Q<GP@!DA!F@T!DN!KM@N@IO!
KMDIO,?W¢N@O?@A<PGO,<<<¡WV„„
<<<M@OPMIN!BDQ@I!Q<GP@!AJM!I@R!F@T!<I?!
PK?<O@N!JMDBDI<G!?D>ODJI<MT!RDOC!BDQ@I!K<DM!<fifl
KMDIO,?W¢N@O?@A<PGO,<<=><¡WV„„!
KMDIO,?W„
```

```
…W£!V¡!Y£!V¡!<<<£!V¡!<=<£!V‰

…W£!/JI@¡!Y£!/JI@¡!<<<£!/JI@¡!<=<£!/JI@‰

…W£!Y¡!Y£!Z¡!<<<£![¡!<=<£!\¡!<><£!]‰

[

WV

…W£!Y¡!Y£!Z¡!<<<£![¡!<=<£!\¡!<><£!]¡!<<=><£!WV‰
```

Thank You

PYTHON PROGRAMMING

SETS AND SET OPERATIONS, FROZENSET

OUTLINE

SETS

- Unordered
- Unindexed
- Unique elements
- Elements cannot be updated, but elements can be removed or added (hence mutable in the sense that items can be added or removed but existing values cannot be updated on the basis of positions)
- List, set, dictionary cannot be elements of a set

Creating set

```
NW˙…W¡X¡<<=><¡,W¡X¡Y„‰

°PNDIB!N@O,„!API>ODJI¢!&HKOT!N@O¡!NX˙…‰!RDGG!>M@<O@!<!?D>ODJI<MT
NX˙N@O,„

°O<F@N!<I!DO@M<=G@!<N!<I!<MBPH@IO!
NY˙N@O,”W¡X¡Y»„

°M@K@<O@?!@G@H@ION!<M@!>JIND?@M@?!JIGT!JI>@
NZ˙…X¡Y¡Z¡X¡W¡Z¡Y‰
KMDIO,NW„
KMDIO,NX„
KMDIO,NY„!
KMDIO,NZ„

065165
…W¡!X¡!,W¡!X¡!Y„¡!<<=><‰
N@O,„
…W¡!X¡!Y‰
…W¡!X¡!Y¡!Z‰
```

Access and Add elements

Since there is no indexing, specific elements cannot be accessed.

```
NW˙…W¡X¡<<=><¡,W¡X¡Y„‰

°<>>@NNDIB!N@O
KMDIO,NW„

°><I!<??!JI@!@G@H@IO
NW¢<??,Y„
KMDIO,NW„

°><I!<??!HPGODKG@!Q<GP@N!
NW¢PK?<O@,<<<¡<=<„
KMDIO,NW„

065165

…W¡X¡<<=><¡,W¡X¡Y„‰
…W¡!X¡!Y¡!,W¡!X¡!Y„¡!<<=><‰
…W¡!X¡!Y¡!,W¡!X¡!Y„¡!<=<¡!<<=><¡!<<<‰
```

Remove elements

```
NW˙…W¡X¡‹<=>‹¡,W¡X¡Y„‰
NW¢?DN><M?,W„
NW¢M@HJQ@,X„
KMDIO,NW„

°M@HJQ@N!<M=DOM<MT!DO@H
KMDIO,NW¢KJK,„„!
KMDIO,NW„
NW¢>G@<M,„
KMDIO,NW„

065165
…‹<=>‹¡!,W¡!X¡!Y„‰
<=>
…,W¡!X¡!Y„‰
N@O,„
```

Basic Operations (+ and * cannot be applied)

```
NW˙…W¡X¡Y‰

KMDIO,W!DI!NW„

KMDIO,W!IJO!DI!NW„

AJM!D!DI!NW£

    KMDIO,D„!


065165

5MP@

’<GN@

W

X

Y
```

Set Operations

```
NW˙…W¡X¡Y¡[‰

NX˙…Z¡[¡\‰

KMDIO,NW¢PIDJI,NX„„

KMDIO,NW°NX„

KMDIO,NW¢DIO@MN@>ODJI,NX„„

KMDIO,NWóNX„

KMDIO,NW¢?DAA@M@I>@,NX„„

KMDIO,NW'NX„

KMDIO,NX¢?DAA@M@I>@,NW„„!!!

KMDIO,NW¢NTHH@OMD>¤?DAA@M@I>@,NX„„

KMDIO,NWûNX„!
```

```
065165
…W¡!X¡!Y¡!Z¡![¡!\‰
…W¡!X¡!Y¡!Z¡![¡!\‰
…[‰
…[‰
…W¡!X¡!Y‰
…W¡!X¡!Y‰
…Z¡!\‰
…W¡!X¡!Y¡!Z¡!\‰
…W¡!X¡!Y¡!Z¡!\‰
```

Built in functions and methods

```
N@OW˙…Z¡X¡ˆ¡WV¡\¡Z¡X‰

°M@K@<O@?!@G@H@ION!<M@!>JPIO@?!JIGT!JI>@
KMDIO,G@I,N@OW„„!
KMDIO,H<S,N@OW„„
KMDIO,HDI,N@OW„„

°M@K@<O@?!@G@H@ION!<M@!>JIND?@M@?!JIGT!JI>@
KMDIO,NPH,N@OW„„
N@OX˙…V‰
N@OY˙…<V<‰
N@OZ˙…V¡W¡X¡Y‰

°M@OPMIN!5MP@!@Q@I!DA!JI@!@G@H@IO!C<N!=JJG@<I!Q<GP@!5MP@
KMDIO,<IT,N@OW„¡<IT,N@OX„¡<IT,N@OY„¡<IT,N@OZ„„

°M@OPMIN!5MP@!JIGT!DA!<GG!@G@H@ION!C<Q@!=JJG@<I!Q<GP@!5MP@
KMDIO,<GG,N@OW„¡<GG,N@OX„¡<GG,N@OY„¡<GG,N@OZ„„
KMDIO,NJMO@?,N@OW¡M@Q@MN@˙5MP@„„

°K<DMN!N@O!@G@H@ION!RDOC!<I!DI?@S!Q<GP@!<I?!M@OPMIN!<N!GDNO
KMDIO,GDNO,@IPH@M<O@,N@OW„„„
```

```
0POKPO
[
WV
X
YV
5MP@!’<GN@!5MP@!5MP@
5MP@!’<GN@!5MP@!’<GN@
”WV¡!ˆ¡!\¡!Z¡!X»
”,V¡!X„¡!,W¡!Z„¡!,X¡!\„¡!,Y¡!ˆ„¡!,Z¡WV„»
```

```
NWˈ…W¡X¡Y‰
NXˈ…Y¡Z¡[‰
NYˈ…[¡\¡]‰
NZˈ…]¡^¡_‰

°K@MAJMHN!DIO@MN@>ODJI!<I?!PK?<O@N!OC@!JMDBDI<G!N@O!
NW¢DIO@MN@>ODJI¤PK?<O@,NX„
KMDIO,›"AO@M!DIO@MN@>ODJI!PK?<O@!NW!<I?!NX!£›„
KMDIO,NW„
KMDIO,NX„

°K@MAJMHN!?DAA@M@I>@!<I?!PK?<O@N!OC@!JMDBDI<G!N@O
NX¢?DAA@M@I>@¤PK?<O@,NY„
KMDIO,›"AO@M!?DAA@M@I>@!PK?<O@!NX!<I?!NY£›„
KMDIO,NX„
KMDIO,NY„

°K@MAJMHN!NTHH@OMD>!?DAA@M@I>@!<I?!PK?<O@N!OC@!JMDBDI<G!N@O
NY¢NTHH@OMD>¤?DAA@M@I>@¤PK?<O@,NZ„
KMDIO,›"AO@M!NTHH@OMD>!?DAA@M@I>@!PK?<O@!NY!<I?!NZ£›„
KMDIO,NY„
KMDIO,NZ„
```

```
065165
"AO@M!DIO@MN@>ODJI!PK?<O@!NW!<I?!NX!£
…Y‰
…Y¡!Z¡![‰
"AO@M!?DAA@M@I>@!PK?<O@!NX!<I?!NY£
…Y¡!Z‰
…[¡!\¡!]‰
"AO@M!NTHH@OMD>!?DAA@M@I>@!PK?<O@!NY!<I?!NZ£
…[¡!\¡!^¡!_‰
…^¡!_¡!]‰
```

NW˙…W¡X¡Y‰

NX˙NW¢>JKT,„

KMDIO,NW„

KMDIO,NX„

NY˙…X¡Y‰

KMDIO,NW¢DN?DNEJDIO,NY„„

KMDIO,NY¢DNNP=N@O,NW„„

KMDIO,NW¢DNNPK@MN@O,NY„„

<u>065165</u>

…W¡!X¡!Y‰

…W¡!X¡!Y‰

’<GN@

5MP@

5MP@

The frozenset

Frozenset can be considered as an ***immutable set***. In general, frozenset() is an inbuilt function that takes any iterable object as argument and freezes them (makes them immutable). Therefore, a frozenset can be a key for dictionary but a set cannot.

Syntax : frozenset(iterable_object)

```
° >JIQ@MON!N@O!DIOJ!<!AMJU@IN@O
N@O˙…W¡X¡Y
‰KMDIO,AMJU@IN@O,N@O„„

° >JIQ@MON!GDNO!DIOJ!AMJU@IN@O
GDNO˙”W¡X¡Y»
KMDIO,AMJU@IN@O,GDNO„„

°AMJU@IN@O!<N!F@T!AJM!?D>ODJI<MT
?D>ODJI<MT˙…AMJU@IN@O,N@O„£W¡!<<=><£X¡!<?@Afl£Y‰
KMDIO,?D>ODJI<MT„
```

```
0POKPO
AMJU@IN@O,…W¡!X¡!Y‰„
AMJU@IN@O,…W¡!X¡!Y‰„
…AMJU@IN@O,…W¡!X¡!Y‰„£!W¡!<<=><£!X¡!<?@A<£!Y‰
```

Thank You