Python Basics cheat Sheet

Variable name:
Variable name in Python can only contain letters (a - z, A - B), numbers or underscores (_).

However, the first character cannot be a number. Hence, you can name your variables user_car or user_car2 but not 2user_car.

variable names are case sensitive so user_car is not the same as user_Car or User_car or User_Car.

Variable name:

Special words can’t be used because these words are built in as a part of python expressions and functions like print, input, if, while, for etc.

These words are (False, class, finally, is, return, None, continue,For, lambda, try, True, def, from, nonlocal, while, and, with, as, elif, if, else, or, yield, assert, import, pass, break, except, in, raise)

Operators types :

Assignment Operators:
give you new value of variable
x = x + 5 can be coded by another way as x + = 5

x = x – 5 can be coded by another way as x - = 5

Logic operators : give you False or True

x > 5 x more than 5 or x >= 5 x more than or equal to 5

x < 5 x less than 5 or x >= 5 x less than or equal to 5

also

x != 5 x not equal to 5

Arithmetic Operators:
Suppose x = 3, y = 2

Addition:

x + y = 5

Subtraction:

x - y = 1

Multiplication:

x*y = 6

Division:

x/y = 1.5

Floor Division:

x//y = 1 (rounds down the answer to the nearest whole number)

Modulus:

x%y = 1 (gives the remainder when 3 is divided by 2)

Exponent:

x**y = 9 (3 to the power of 2)

Python data types:

Integers:

integers are numbers with no decimal parts, such as -3, -1, -2

, 0, 6, 10 etc.

To declare an integer in Python, simply code as following :

Variable_name = interger value

Examples:

user_car_speed = 100

car_ number_plate= 123569

Python data types:Floats
Floats are numbers that have decimal parts, such as 1.56, -0.0056,

10.015

To declare a float in Python, you will code as following:

Variable_name = float value

Examples:

user_car_speed = 100.55

User_height = 1.72

Python data types: strings
String are composed of elements like letters, numbers, special characters to build a text. To declare a string, you can either use variable_name = ‘string’ (single quotes)

Variable_name = “string” (double quotes)

Example:

user_car_speed = ‘100’ or user_car_speed = “100”

user_car_color = ‘red !?’ or user_car_color = “red !?”

Casting in python:
Casting in python is converting one data type to another data type.

Using build_in python functions int(), float(), and str()

Examples:

user_car_speed = “ 100.55 ”

float (user_car_speed) = 100.55

int(float (user_car_speed)) = 100

```
int(user_car_speed) = 100
```

String functions in python:
Format() function :

```
>>>name = “Alexander”

>>>car_color = “red.”

>>>print(“hello world, my name is”+ “ ”+name+“ ”+“,and my car color is”

+ “ “+car_color)

hello world, my name is Alexander ,and my car color is red

msg = “hello world, my name is {} , and my car color is {}.format(name, car_color)

>>> print(msg)

hello world, my name is Alexander ,and my car color is red.
```

String len() function:

```
>>> print(len(msg))

58

>>> print(len(name))

9

>>> print(len(name[0:4]))

4

>>> Print(Name[0:4])

Alex
```

Used to count the entire string or specific character of a string

```
>>>‘This is user car’.count(‘r’)

2
```

In order to count from index 5 to end of string

```
>>>‘This is user car’.count(‘r’, 5)

2
```

In order to count from index 5 to 12

```
>>> ‘This is user car(‘s’, 5, 12 )

1
```

In orde count ‘T’. There’s only 1 ‘T’ as the function is case sensitive:

Example:

```
>>>‘This is user car’.count(‘T’)

1

>>>‘This is user car’.count(‘u’)

1

>>>‘This is user car’.count(‘U’)

0
```

endswith () and startswith() functions:
this function Return True if the string ends with the specified *suffix*, otherwise return False and it is case-sensitive function.

Example:

In order to check if a string contain specific suffix like car

```
>>>‘user_car’.endswith(‘car’)

True

# check from index 4 to end of string

>>>“user_car’.endswith(‘car’, 4)
```

True

To check from index 5 to 7

```
>>>‘user_car’.endswith(‘car’, 5, 7)
```

```
False
```

To check from index 4 to 8

```
>>>“user_car’.endswith(‘car’,4, 8)
```

```
True
```

In order to use a tuple of suffixes as we check from index 4 to 8

```
>>>“user_car’.endswith((‘car’, ‘ar’), 4, 8)
```

```
True
```

Startswith() like endwiths() but used for bega

Find() string function:
Return the index in the string where the first occurrence of the substring *sub* is found

Examples:

In order to check specific character in a string

```
>>>‘This is user_car’.find(‘i’)
```

```
2
```

In order to check from index 2 to end of a string

```
>>>‘This is user_car’.find(‘s’, 2)
```

```
3
```

In order to check from index 8 to 12

```
>>>‘This is user_car’.find(‘s’, 8,12 )

9
```

In order to check special character which is not found in string

.find() function is like .index() function but differ in the next example:

```
>>> 'This is user_car'.find(‘d’)

-1

>>> 'This is user_car'.index(‘d’)

ValueError
```

islower () and isupper() functions:
islower() return True if all characters in the string are lowercase and return False otherwise.

[Example]

```
>>>‘abc’.islower()

True

>>>‘ABc’.islower()

False

>>>‘ABC’.islower()

False
```

isupper() :

return True if all characters in the string are uppercase and return False otherwise.

Examples:

```
>>>‘ABC’.isupper()
```

```
True

>>>‘AbC’.isupper()

False

>>>‘abc’.isupper()

False
```

lower() and upper() functions:
Lower (): the function that convert string case into lower case.

Example:

```
>>>‘This is User Car’.lower()

this is user car
```

Example:

upper (): the function that convert string case into upper case.

```
>>>‘This is User Car’.upper ()

THIS IS USER CAR
```

replace() string function:
This function return a copy of the string with all occurrences of substring old replaced by new.

[Example]

```
>>>‘This is user car’.replace(‘user’, ‘my’)

This is my car
```

In order to replace a character many times:

```
>>>‘This is user car’.replace(‘s’, ‘c’, 3)
```

'Thic ic ucer car'

strip() string functions:
This function return a copy of the string with the starting orending character removed but If character is not provided, whitespaces will be removed.

Example:

```
>>>‘ This is my car’.strip()

'This is my car'

>>>‘ This is my car’.strip(‘p’)

'This is a my car'

>>>‘This is my car’.strip(‘r’)

‘This is my ca’
```

Print and Input functions:
We take and practice print() function which is so simple, now we will take about input() function which store variable data as string

Example:

```
>>> user_name = input(“Please enter your name”)

Please enter your name

>>> Alex

>>> print(user_name)

Alex
```

How to deal with string special characters in python?
In order to print string without errors we should use back slash

Character to prevent interfere of these characters with string quotes.

Example:

```
>>> msg1= “I’m”

>>> msg1= “I\’m”

>>> msg2= “I can’t”

>>> msg2 =" can\’t”

>>> print(msg1+” “ +msg2)

I’m can’t
```

Lists in python : []

Lists are a collection of data values stored as a list and values are separated by a comma.

To declare a list, you code as following :

We use square brackets [ ] when declaring a list.

list_name = [Multiple values are separated by a comma]

Example:

```
user_cars_colors = [‘red’, ‘yellow’, ‘black’, ‘blue’]
```

List commands in python

```
user_cars_colors = [‘red’, ‘yellow’, ‘black’, ‘blue’]
```

To assign user_cars_colors (from index 1 to 3) and print

```
user_cars_colors = user_cars_colors[1:3]

print (user_cars_colors)

>>>You’ll get [yellow, black]
```

In order to modify the second item in List and print the updated list

```
user_cars_colors[1] = ‘green’

print(user_cars_colors)

>>> [‘red’, ‘green’, ‘black’, ‘blue’]
```

To append a new item to user_cars_colors and print the list :

```
user_cars_colors.append(“white”)

print(user_cars_colors)

>>>[‘red’, ‘green’, ‘black’, ‘blue’, ‘white’]
```

To remove the fifth item from user_cars_colors and print the list :

```
del user_cars_colors [4]

print(user_cars_colors)

>>>[‘red’, ‘green’, ‘black’, ‘blue’]
```

len(): returns how many elements are in a list.

```
Users_cars_speeds = [100, 50, 30, 20,120]

len(Users_cars_speeds)

>>> 5
```

max() returns the greatest element of the list.

```
Users_cars_speeds = [100, 50, 30, 20,120]

max(Users_cars_speeds)

>>> 120
```

min(): returns the smallest element in a list.

```
Users_cars_speeds = [100, 50, 30, 20,120]
```

```
min(Users_cars_speeds)

>>> 20
```

sorted() returns a copy of a list ordered from smallest to largest

```
Users_cars_speeds = [100, 50, 30, 20,120]

sorted(Users_cars_speeds)

>>> 120, 100, 50, 30, 20
```

Sum():

returns the sum of the elements in a list.

```
>>> Users_cars_speeds = [100, 50, 30, 20,120]

>>>sum(Users_cars_speeds)

320
```

pop():

removes the last element from a list and returns it.

```
>>> Users_cars_speeds = [100, 50, 30, 20,120]

>>> Users_cars_speeds.pop

120
```

Using range( ) in list:

```
>>> x = range(5)

>>> print(x)
[0,1,2,3,4]
```
Zero = x[0] *equals 0, lists are 0-indexed*
One = x[1]
Two = x[2]

Three = x[3]

Four = x[4] fifth element of the x list of range(5).
Four = x[-1] fifth element of the x list of range(5).
>>>

Tuples : ()
Tuples are just like lists, but we cannot modify their data values.

Example:

Year_monthes = (“January”, “February”, “March”, “April”, “May”, “June”, “July”, “August”, “September”, “October”, “November”, “December”)

In order to access specific values of a tuple use their indexes like what we do with a list.

so, year_monthes [0] = “January”, & year_monthes [-1] = “December”

Del() tuple function:

This function will delete the whole tuple

[Example]

year_monthes = (“January”, “February”, “March”, “April”, “May”, “June”, “July”, “August”, “September”, “October”, “November”, “December”)

del year_monthes

print (year_monthes)

=> NameError: name ' year_monthes ' is not defined

In use for tuples:

This function will check if an item is in a tuple

[Example]

year_monthes = (“January”, “February”, “March”, “April”, “May”, “June”, “July”, “August”, “September”, “October”, “November”, “December”)

‘1’ in year_monthes

=> False

‘January’ in year_monthes

=> True

len( ) tuple function:

This function will calculate number of items in tuple.

[Example]

year_monthes = (“January”, “February”, “March”, “April”, “May”, “June”, “July”, “August”, “September”, “October”, “November”, “December”)

print (len(year_monthes))

>>> 12

Addition of tuples:

year_monthes = (“January”, “February”, “March”, “April”, “May”, “June”, “July”, “August”, “September”, “October”, “November”, “December”)

print (year_monthes +(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12))

>>>(“January”, “February”, “March”, “April”, “May”, “June”, “July”, “August”, “September”, “October”, “November”, “December”, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)

multiplication of tuples:

year_monthes = (“January”, “February”, “March”, “April”, “May”, “June”, “July”, “August”, “September”, “October”, “November”, “December”)

print (year_monthes * 2)

>>>(“January”, “February”, “March”, “April”, “May”, “June”, “July”, “August”, “September”, “October”, “November”, “December”, “January”, “February”, “March”, “April”, “May”, “June”, “July”, “August”, “September”, “October”, “November”, “December”)

Dictionary : {}

Dictionary is group of pairs of data values each pair have two parts

Dictionary key and data value {key1:value1, key2:value 2, etc….}

Example:

```
>>> User_name_and_car_color = { “David”:”red”, “John”:”green”, ”Sara”: “yellow”}
```

In order to declare a dictionary we use dict() method

```
>>> User_name_and_car_color = dict(David = red, John = green, Sara = yellow)

{“David”:”red”, “John”:”green”, ”Sara”: “yellow”}
```

In order to add a new item and print the updated dictionary:

```
>>>user_name_and_car_color[“Kate”] = “black”

>>>print(user_name_and_car_color)

{“Kate”, “black”, “David”:”red”, “John”:”green”, ”Sara”: “yellow”}
```

In order to remove the item with key = “Kate” and print dictionary:

```
>>>del user_name_and_car_color [“Kate”]

>>>print(user_name_and_car_color)

{“David”:”red”, “John”:”green”, ”Sara”: “yellow”}
```

Example:

```
>>>Print(user_name_and_car_color)

{“David”:”red”, “John”:”green”, ”Sara”: “yellow”}
```

In order to print the item with key = “David” code as following:

```
>>>print(user_name_and_car_color[“David”]) red
```

In order to print the item with key =Sara code as following:

```
>>> print(user_name_and_car_color[“Sara”])

yellow
```

In order modify the item with key = John and print the dictionary

```
>>> user_name_and_car_color[“John”] = “white”

>>>print(user_name_and_car_color”David”)

red
```

Dictionary .get() :

returns a value for the given key If the key is not found, it’ll return the keyword None.

Example:

```
user_name_and_car_color = {“David”:”red”, “John”:”green”, ”Sara”: “yellow”}

print(user_name_and_car_color)

>>>{“David”:”red”, “John”:”green”, ”Sara”: “yellow”}

print(user_name_and_car_color. get(“John”))

>>>green

user_name_and_car_color = {“David”:”red”,“John”:”green”, ”Sara”: “yellow”}

>>>print(user_name_and_car_color)

{“David”:”red”, “John”:”green”, ”Sara”: “yellow”}

print(user_name_and_car_color. get(“kate”))

None
```

Dictionary .del( ):

It delete the whole dictionary.

Example:

```
>>> user_name_and_car_color = {“David”:”red”, “John”:”green”, ”Sara”: “yellow”}

>>> user_name_and_car_color.del()

>>>print (user_name_and_car_color)

NameError: name 'dic1' is not defined
```

Dictionary .clear( ):

It removes all elements of the dictionary, returning an empty dictionary

Example:

```
>>> user_name_and_car_color = {“David”:”red”, “John”:”green”, ”Sara”: “yellow”}

>>> user_name_and_car_color.clear()

>>>print (user_name_and_car_color)

{ }
```

Dictionary In( ):

We use it to check if an item is in a dictionary

Example:

```
>>>user_name_and_car_color = {“David”:”red”, “John”:”green”, ”Sara”: “yellow”}
```

Use it on the key

```
>>>David in user_name_and_car_color

True

>>>Kate in user_name_and_car_color
```

False

Use It on the value:

Example:

user_name_and_car_color = {“David”:”red”, “John”:”green”, ”Sara”: “yellow”}

‘red’ in user_name_and_car_colo.values()

=> True

‘black’ in user_name_and_car_colo.values()

=> False

Dictionary .items( )

We use it to return a list of dictionary’s pairs as tuples

[Example]

user_name_and_car_color = {“David”:”red”, “John”:”green”, ”Sara”: “yellow”}

user_name_and_car_color.items()

>>> dict_items([(“David”:”red”), (“John”:”green”), (”Sara”: “yellow”)])

Dictionary .len( ):

We use it to calculate the number of items in a dictionary

Example:

>>>year_monthes = {“January” : 1, “February” : 2, “March” : 3, “April”: 4, “May”: 5, “June”: 6, “July”: 7, “August”: 8, “September”: 9, “October”: 10, “November”: 11, “December”: 12 }

>>> print (len(year_monthes))

12

```
>>>
```

Dictionary . update( ):

We use this to Add one dictionary’s key-values pairs to another. Duplicates are removed.

Example:

```
>>> year_monthes = {“January” : 1, “February” : 2, “March” : 3, “April”: 4, “May”: 5, “June”: 6}

>>> second_six_year_monthes = {“July”: 7, “August”: 8, “September”: 9, “October”: 10,
“November”: 11, “December”: 12}

>>> year_monthes.update(second_six_year_monthes)

>>> print (year_monthes)

{“January” : 1, “February” : 2, “March” : 3, “April”: 4, “May”: 5, “June”: 6, “July”: 7, “August”: 8,
“September”: 9, “October”: 10, “November”: 11, “December”: 12 }

>>>print second_six_year_monthes

{“July”: 7, “August”: 8, “September”: 9, “October”: 10, “November”: 11, “December”: 12}
```

So there is no change in second_six_year_monthes dictionary.

Dictionary . values( )

This will returns list of the dictionary's values

Example:

```
>>> year_monthes = {“January” : 1, “February” : 2, “March” : 3, “April”: 4, “May”: 5, “June”: 6,
“July”: 7, “August”: 8, “September”: 9, “October”: 10, “November”: 11, “December”: 12 }

>>>year_monthes.values()

dict_values([‘1', ‘2‘, ‘3', ‘4‘, ‘5', ‘6‘, ‘7', ‘8‘, ‘9', ‘10‘, ‘11', ‘12‘])
```

Dictionary . keys( )

We use it to returns list of the dictionary's keys

[Example]

>>> year_monthes = {“January” : 1, “February” : 2, “March” : 3, “April”: 4, “May”: 5, “June”: 6, “July”: 7, “August”: 8, “September”: 9, “October”: 10, “November”: 11, “December”: 12 }

>>> year_monthes.keys()

dict_keys([“January”, “February”, “March”, “April”, “May”, “June”, “July”, “August”, “September”, “October”, “November”, “December”])

Sets:

set is data structure, which represents a collection of*specific* elements:
set = set()
set.add(a)
set.add(b)
set.add(c)
x = len(set) print(x) >>>x= 3
y = d **in** set print(y) >>>y= False
z = c **in** set print(z) >>>z= False

Why it is preferred to use sets?

1. It is a very fast code operation on sets than lists.
2. set is it more appropriate than a list, If we have a large collection of items that we want to use for elements presence in set test:

>>> x = [1,2,3,4,5,6,7,8,9,10]

>>>print(1 in set(x), 15 in set(x),20 in set(x))

(True, False, False)

Types of Exception & Cause of Error:
AssertionError : Raised when `assert` statement fails.

AttributeError : Raised when attribute assignment or reference fails.

EOFError : Raised when the `input()` functions hits end-of-file condition.

FloatingPointError : Raised when a floating point operation fails.

GeneratorExit : Raise when a generator's `close()` method is called.

ImportError : Raised when the imported module is not found.

IndexError : Raised when index of a sequence is out of range.

KeyError : Raised when a key is not found in a dictionary. KeyboardInterrupt Raised when the user hits interrupt key (Ctrl+c or delete).

MemoryError : Raised when an operation runs out of memory.

NameError : Raised when a variable is not found in local or global scope.

NotImplementedError : Raised by abstract methods.

OSError : Raised when system operation causes system related error.

OverflowError : Raised when result of an arithmetic operation is too large to be represented. ReferenceError Raised when a weak reference proxy is used to access a garbage collected referent.

RuntimeError : Raised when an error does not fall under any other category.

StopIteration : Raised by `next()` function to indicate that there is no further item to be returned by iterator.

SyntaxError : Raised by parser when syntax error is encountered.

IndentationError : Raised when there is incorrect indentation.

TabError : Raised when indentation consists of inconsistent tabs and spaces.

SystemError :Raised when interpreter detects internal error.

SystemExit : Raised by `sys.exit()` function.

TypeError : Raised when a function or operation is applied to an object of incorrect type.

UnboundLocalError : Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.

UnicodeError : Raised when a Unicode-related encoding or decoding error occurs.

UnicodeEncodeError : Raised when a Unicode-related error occurs during encoding.

UnicodeDecodeError : Raised when a Unicode-related error occurs during decoding.

UnicodeTranslateError : Raised when a Unicode-related error occurs during translating.

ValueError : Raised when a function gets argument of correct type but improper value.

ZeroDivisionError : Raised when second operand of division or modulo operation is zero.

Python functions:
Function: are pre-written lines of code that do a specific task.

Function: are composed of:

1. Definition code line.

2. Argument.

3. Retrun or print or yield or result making lines of code.

Python functions:

We define our own functions in Python and reuse them in the same program as many times as we need.

def function_name(parameters):

lines of code of the function should do

return something

Variable Scope:

There two types of variables according to variable scope:

Local variable:

When the variable located and assigned to data value inside the function only, so it will affect that data value of the that variable inside that function only.

Global variable:

When the variable located and assigned to data value outside the function, so it will affect that data value of the that variable inside and out side that function.

Python functions examples:

1.Function That returns a value according to its argument :

```
>>>def rectangle_area(width, length)

area = width * length

return area

>>>def rectangle_area(6, 9)

Result :54

>>>def rectangle_area(4, I5)

Result : 20
```

2. Function that returns Boolean value :

Example 1:

```
def is_even(x):
if x % 2 == 0:
return True
else:
return False
>>>def is_even(10)

Result: True
```

2. Function that returns Boolean value:

Example 2:

```
def is_odd(y):
if y % 2 == 0:
return True
else:
```

```
return False
>>>def is_odd(8)
```

Result: False

3.Function that prints specific data:

Example 1:

```
def countdown(x):
while x> 0:
print n
x = x-1
print “Stop this is the end!"
>>>def countdown(2)
```

Result: 2

1

Stop this is the end!

3. Function that print specific data:

Example 2:

```
def is_even_or_odd(n):
if n % 2 == 0:
return “this number is even”
else:
return “this number is odd”
>>>def is_even_or_odd(2)
```

Result: this number is even

```
>>>def is_even_or_odd(7)
```

Result: this number is odd

If statement:
If statement is commonly used in programs coded in python and the strcture of if statement as follow:

```
if condition = value:

do something

elif condition = value:

do something

else:

do some thing

If name == “Alex”:

print(“Hi Alex”)

elif name == “Sara”:

print(“Hi Sara”)

elif name == “kate”:

print(“Hi Kate”)

else:

print(“Hi everybody”)

distance = input(“what is the distance of this road?”)

>>> what is the distance of this road?

25

msg= “this road 11 if distance == 20 else “this is another road”

>>>Print (msg)
```

we observe that user input distance = 25

So it will print :

this is another road

For Loop:

The for loop Looping through an iterable executes a block of code repeatedly until the condition in the for statement is no longer valid.

an iterable In Python refers to anything that can be looped over, such as a string, list or tuple.

The syntax for looping through an iterable is as follows:

```
for i in iterable:

print (i)
```

Example:

```
colors = [‘red', ‘yellow', ‘green', ‘blue‘, black]

for color in colors:

print (color)
```

1. we first declare the list of colors and give it the members ‘red', ‘yellow', ‘green', ‘blue‘, black.

2. Next the statement for color in colors, so the program loops through the colors list and assigns each member in the list to the variable color.

If you run the program, you’ll get

```
red

yellow

green

blue

black
```

the index of the members in the list can be displayed by use the enumerate() function.

```
for index, color in enumerate(colors):

print (index, color)

>>>0 red

1 yellow

2 green

3 blue

4 black
```

Looping through a sequence of numbers

Example :

loop through a string data.

```
msg = ‘Hi’

for i in msg:

print (i)

>>>H

i
```

range() function:

if start is not given, the numbers generated start from zero.

For instance,

range(5) will generate the list [0, 1, 2, 3, 4]

range(3, 10) will generate [3, 4, 5, 6, 7, 8, 9] range(4, 10, 2) will generate [4, 6, 8]

the range() function works inside a for statement:

Example:

```
for i in range(3):

print (i)

>>>0

1

2
```

break: using break in for loop:

sometimes we want to stop the loop when meet specific certain condition so we use the break.

Example:

```
a = 0

for i in range(10):

a+= 2

print (‘i = ’, i, ‘, a= ’, a)

if a== 4:

break
```

we will get:

```
i = 0 , a = 2

i = 1 , a= 4
```

Without the break, the coder will loop from i = 0 to i = 10 because we used the function range(10).

However with the break keyword, the program ends prematurely at i = 2. This is because when i = 2, j reaches the value of 6 and the break keyword causes the loop to end.

continue: using continue in for loop:

We use continue to skip specific step in the iteration process.

Example:

```
for num in range(3)

if num == 1:

continue

print('Current number :', num)
```

Result: Current number : 0

Current number : 2

While Loop

while loop repeatedly executes code which inside the loop while specific condition remains valid.

Composition of while loop is as follow:

```
while condition is true:

do something
```

Firstly we declare a variable to function as a loop counter(variable counter)

The condition in which the while statement will evaluate the value of counter to determine if it something more than or less than, so the loop

will be executed.

Example:

```
>>>variable_count= 3

while variable_count > 0:

variable_count = variable_count - 1

print ("Variable_count =" , variable_count)

variable_count = 3

variable_count = 2

variable_count = 1
```

Continue: using continue in while loop:

We use continue to skip specific step in the iteration process of while loop.

Example:

```
num = 3

while num > 0:

num -= 1

if num == 1:

continue

print('Current number:', num)

Result: Current number : 0

Current number : 2
```

Zip( ) iterator examples:

```
print(list(zip(['red', 'yellow', 'green'], [1, 2, 3])))
```

```
>>>[('red ', 1), ('yellow ', 2), ('green', 3)]
colors = ['red', 'yellow', 'green']

nums = [1, 2, 3]

for color, num in zip(colors, nums):

print("{}: {}".format(color, num))

'red’: 1

'yellow': 2

'green': 3
```

Zip( ) iterator +* examples for lists : unzip nested lists

Eaxmple 3:

```
>>>colors = ['red', 'yellow', 'green']

>>>nums = [1, 2, 3]

>>>colors_list = [colors, nums]

>>>for colors, nums in zip(*colors_list):

print(colors, nums)

red 1

yellow 2

green 3
```

Zip( ) iterator+ * example for tuples: unzip nested tuples

```
>>>colors = ('red', 'yellow', 'green')

>>>nums = (1, 2, 3)
```

```
>>>colors_list = (colors, nums)

>>>for colors, nums in zip(*colors_list):

print(colors, nums)

red 1

yellow 2

green 3
```

enumerate( ) iterator: for lists:

It returns an iterator of tuples containing indices and values of a list. use this when you want the index along with each element of an iterable in a loop.

```
>>>letters = ['a', 'b', 'c', 'd', 'e']

>>>for i, letter in enumerate(letters):

print(i, letter))
0 a
1 b
2 c
3 d
4 e
```

enumerate( ) iterator:for tuples

Example 2:

```
letters =('a', 'b', 'c', 'd', 'e‘)
for i, letter in enumerate(letters):
print(i, letter)
0 a
1 b
2 c
3 d
4 e
```