EG1007 Introduction to Programming Tutorial 3
Lists and for loops: Core practices

1) Working with Lists

A list can be expressed by enclosing the list of objects with square brackets, and separate each item in the list with commas. In IDLE Shell, create lists by typing the following commands

```
>>> my_vector = [1,2,3,4,5,6,7,8,9]
>>> my_string_set = ["I","like","Python","I","want","to","learn","more"]
>>> my_letter_list= list("Python Programming")
```

The last one uses list() function to convert a string to a list of characters. You can use print() to show the details.

Lists are ordered with index. From the first element forward, they are indexed as 0, 1,2,…. Python also provides another index system from the last element backward, they are indexed as -1, -2, … The following table lists some statements, what are the output of these statement?

print(my_vector[4])	5
print(my_vector[9])	Error, index out of range
Pring(my_vector[-4])	6
print(my_string_set[0])	I
print(my_string_set(3))	I
print(my_string_set[-3]+my_letter_list[-3])	toi
print(my_string_set[3]==my_string_set[-5])	True

my_vector[2] = [31, 32]
print(my_vector[2][1])
	32

del my_vector[2] print(my_vector[2])	4

my_vector.append(10)
print(my_vector[9])
	10

type(my_vector[2])
my_vector[2] = “two”
type(my_vector[2])
	int
	str

sub_vector = my_vector[2:5]
print(sub_vector[1])
print(sub_vector[3])
print(sub_vector)	

Note:

- del is to delete an element in the list

- append() adds one element to the ***end*** of the list.
- Element of a list can be another list (nested lists). See 8th row in the table
- The data types of elements can be different. See 11th row in the table
- When slicing the lists to form a sub-lists, sub-lists = list_name[startindex:endindex]. The elements in list_name with index >= startindex and < endindex will form the new lists.

2) In addition to while loop, Python provides for loop, which is useful for iterating over a collection of objects, such as a list. In general, for loop is described as

```
for variable in collections:
        …
        … for loop code block
        …
```

In a for loop, in assigns each element to a variable at each step. In the shell, type the following example,

```
>>> words = ['I','am','learning','EG1007','in','OTLT']
>>> for word in words:
...     print(word)
...
...
I
am
learning
EG1007
in
OTLT
```

variable **word = 'I'** in the first iteration, **word = 'am'** in the second iteration. In practices, you may know exactly the range of numbers in the iteration. In the previous tutorial about the while loop, we have used the following example,

```
#example of while loop

n = 10
while n > 0:    #set the condition, if n > 0, execute the code block below
    print(n)    #print the value of n
    n -= 1      #n = n - 1, counting down
                #after this, move back to the beginning of the loop, i.e.
                #   to check whether n > 0, if yes, continue the loop, otherwise, stop
print("Happy New Year")
```

Actually, the value of n in the loop ranges from 10 to 1 with an interval of -1. In Python, we can use range(10,0,-1). Consequently, the above loop can be replaced by a for loop below. Type this in the shell to see whether you can get the same outcome as the above while loop.

```
>>> for n in range(10,0,-1):
...     print(n)
...
...
```

In general, range(startNum, endNum, interval) would give you a sequence of numbers in the range [startNum, endNum) with a pacing of interval. *Note: endNum does not include in the sequence. This is why in the above example, n = 0 does not occur.*

When introducing while loop, we discuss the use of break to exit the loop. Here we further introduce continue and pass in the loop. In the shell, type the following code

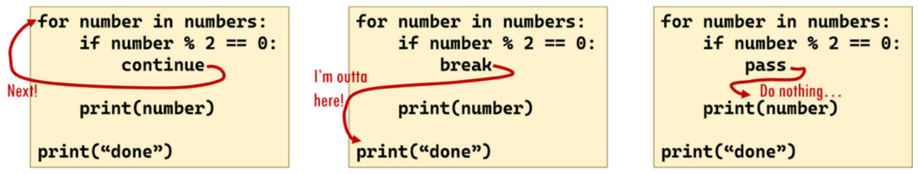
```
>>> numbers = range(0,10)
>>> for number in numbers:
...     print(number)
```

```
>>> for number in numbers:
...     if number%2 == 0:
...         continue
...     print(number)
```

```
>>> for number in numbers:
...     if number%2 == 0:
...         break
...     print(number)
```

```
>>> for number in numbers:
...     if number%2 == 0:
...         pass
...     print(number)
```

The difference output in the above code with continue, break and pass, is illustrated in the following chart,



```
for number in numbers:
    if number % 2 == 0:
        continue

    print(number)

print("done")
```

```
for number in numbers:
    if number % 2 == 0:
        break

    print(number)

print("done")
```

```
for number in numbers:
    if number % 2 == 0:
        pass
    print(number)

print("done")
```

3) With the preparation in 2), analyse the outcome of the following codes. Put your answer in the right column. After doing so, type the code in the left column in a Python script. Run the script to check your answer.

Python Code	Outcome

`numbers= [5, 1, 3]`
`total  = 0`
`for number in numbers:`
`    total += number`
`print(total)`
	9

`numbers= range(1,7)`
`total = 0`
`for number in numbers:`
`    if number % 2 == 0:`
`        total = total + number`
`print(total)`	

`numbers = [3,2,1,0,1,2,3]`
`total = 0`
`for number in numbers:`
`    if number <= 0:`
`        break`
`    total = total + number`
`print(total)`	

`matrix = [[1,2,3],[4,5,6],[7,8,9]]`
`total  = 0`
`for row in matrix:`
`    row_total = 0`
`    for cell in row:`
`        row_total = row_total + cell`
`    print(row_total)`
 	
`    total += row_total`
 	
`print(total)`	

```
total = 0
for number in range(0,10,2):
    total += number
print(total)
```

```
for number in range(1,9):
    print(number,end = " ")
print()
```

```
for row in range(1, 10):
    for col in range(1, row+1):
        print(col, end=" ")
    print("*")
```

```
numbers = [5, 3, 2, 1, 4, 6]
total = 0
for number in numbers:
    if number % 2 == 0:
        continue

    total += number

print(total)
```

```
numbers = [5, 3, 2, 1, 4, 6]
total = 0
for number in numbers:
    if number % 2 == 0:
        break

    total += number

print(total)
```

```
numbers = [5, 3, 2, 1, 4, 6]
total = 0
for number in numbers:
    if number % 2 == 0:
        pass

    total += number

print(total)
```

```
matrix = [[1, 0, 2], [3, 1, -4], [2, -2, 2]]
for row in matrix:
    row_total = 0
    for cell in row:
        row_total += cell

    if row_total <= 0:
        continue

    print(row_total)
```

```
matrix = [[1, 0, 2], [3, 1, -4], [2, -2, 2]]
for row in matrix:
    row_total = 0
    for cell in row:
        if cell <= 0:
            continue
        row_total += cell

    print(row_total)
```

4) Using for loop to complete the following task

i) Giving a value of n and calculate $\sum_{i=1}^{n} i$, i.e. 1 + 2 + 3,…,+n

ii) Define a function to calculate $\sum_{i=1}^{n} i$ with an input variable n. Call the function you defined with different value of n

iii) Define a function to calculate n!, the fractional of n, i.e. 1*2*3*…*n, with the input variable of n.

iv) Using the function defined in iii) to calculate e using

$$e = 1 + 1 + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{n!}$$

within for loop, evaluate the error $\varepsilon = \frac{1}{n!}$. When the error is smaller than 1e-6, break the loop.

v) Print out Fibonacci Sequence, 1, 1, 2, 3, 5, 8, … The number at i-th (i = 4, …) position is the sum of numbers at (i-1) and (i-2)-th positions. You can set the size of the sequence and specify the first and 2nd numbers are all 1. Break the loop when the number is greater than 100.

vi) Print the following image in screen

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
```