BU CodeBreakers 2017

Python IV: User Inputs, Loops

Today’s Schedule

09:00-10:00	Python IV
10:15 - 11:15	Exercise IV

We expect everyone get their lunch and meet with us in the Back room @ 11:45

12:00 - 01:45	Climbing
02:00 - 03:00	Q&A Session

Taking input from user

Command: input (<prompt string>)

```
>>> x = input ('Enter Stuff: ')
```

Accepted input types:

- Int
- Float
- String
- List
- Boolean

```
>>> print x
```

Problem

“Generate 100 random integers that are less than 100”

Problem

“Generate 100 random integers that are less than 100”

Solution I

```
1.   print ( int (random()*100) )
2.   print ( int (random()*100) )
3.   print ( int (random()*100) )
4.   ...

...

100. print ( int (random()*100) )
```

Problem

“Generate 100 random integers that are less than 100”

Solution II

Use **for** loop

```
for i in range (100) :
        print ( int (random()*100) )
```

Functions?

range, int, random, print

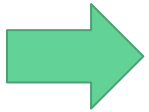
Loops

- Piece of same-code that executes multiple times in a program. Example: for loop

```
1.   print ( int (random()*100) )
2.   print ( int (random()*100) )
3.   print ( int (random()*100) )
4.   ...

...

100. print ( int (random()*100) )
```



```
1.   for i in range (100) :
2.           print ( int (random()*100) )
```

Loops

- Piece of same-code that executes multiple times in a program. Example: for loop

```
1.   print ( int (random()*100) )
2.   print ( int (random()*100) )
3.   print ( int (random()*100) )
4.   ...

...

100. print ( int (random()*100) )
```

```
1.   for i in range (100):
2.           print ( int (random()*100) )
```

Loop examples

```
>>> for i in [1, 3, 5, 7]:
        print (i)

>>> for x in [7]*4:
        print (x)

>>> for c in ‘CodeBreakers’:
        print (c)
```

Syntax: **for** <variable> **in** <sequence>:

Do Stuff

range() Functions

```
>>> print range(10)
```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

What does range(n) do?

➔ Makes a list of integers from 0 to n-1

```
>>> print range(-1, 6, 2)
```

[-1, 1, 3, 5]

What does range(x, y, s) do?

➔ Makes a list of integers from *x* to *y-1* with step of *s* => range(start, stop, step)

Optional!

For **loop**

Start

```
1.  for x in [2, 4, 6, 8]:
2.      print 'x is: ', x
3.      print 'still in loop'

4.  print 'Done'
```

End

element-based looping

=

```
1.  my_list = [2, 4, 6, 8]
2.  for i in range(len(my_list)):
3.      print 'x is: ', my_list[i]
4.      print 'still in loop'

5.  print 'Done'
```

index-based looping

while **loop**

Syntax:

1. **while** <condition>**:**
2. Do stuff_1
3. Do stuff_2
4. Out of loop

condition → **True**

condition → **False**

1. while True:
2. Do stuff_1
3. Do stuff_2
4. Out of loop

True case

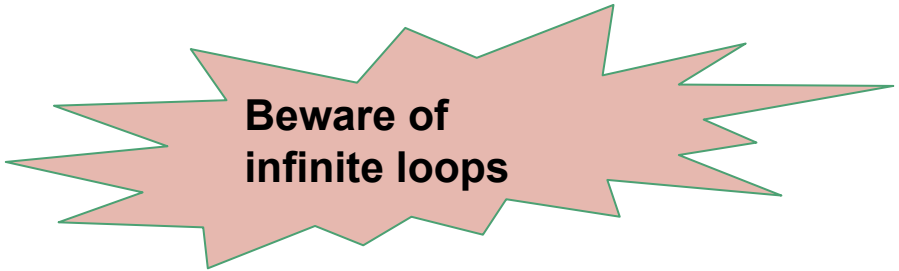
1. while False:
2. Do stuff_1
3. Do stuff_2
4. Out of loop

False case

while examples

```
>>> x = [2, 4, -1, 5]
>>> i = 0
>>> while x[i] > 0:
        print x[i]
        i = i+1
>>> print 'Done'
```

```
>>> i = 5
>>> while i > 4.5:
        print 'looping'
>>> print 'Done'
```



Loops Forever!!!! Condition is always true!!!

What does this function do?

```
def my_function( my_string ):
    n = 0
    for c in my_string:
        if c in 'aeiouAEIOU':
            n += 1
    return n
```

```
>>> my_function('Hello')
```

c	n
	0
H	0
e	1
l	1
l	1
o	2

Output → n → 2

What does this function do?

```
def my_function( n ):
    while n != 1:
        if n%2 == 0:
            n = n/2
        else:
            return False
    return True
```

```
>>> my_function(8)

------
n
------
8
4
2
1
```

Output → True

```
>>> my_function(12)

------
n
------
12
6
3
```

Output → False

Complete this function:

```
#Function to return minimum number in a list
def min_list( L ):
    min_val = L[0]
    for x in L:
        #strategy?
        if <condition?> :
            <statement?>
    return min_val
```

Complete this function:

```
#Function to return minimum number in a list
def min_list( L ):
    min_val = L[0]
    for x in L:
        #replace min_val with smaller element
        if x < min_val :
            min_val = x
    return min_val
```

```
>>> min_list( [2, 3, -1, 4] )
```

x	min_val
	2
2	2
3	2
-1	-1
4	-1

Output → min_val → -1

Exercise

- Write function to return **True** if *input n* is a prime number and **False** otherwise

Solution:

```
def isPrime( n ):
    number_divisors = 0
    for i in range(1, n):
        if n%i==0:
            number_divisors += 1
    if number_divisors > 1:
        return True
    else:
        return False
```

>>> isPrime(6)

i	number_divisors
	0
1	1
2	2
3	3
4	3
5	3

Output → False

>>> isPrime(5)

i	number_divisors
	0
1	1
2	1
3	1
4	1

Output → False

1. exercise4_level1.py
2. exercise4_level2.py