New Topic

Repetition and Loops

Additional Python constructs that allow us to effect the (1) order and (2) **number of times** that program statements are executes.

These constructs are the

1. **while loop** and
2. **for loop**.

Recall:

1. Input
2. Output
3. Memory
4. Arithmetic – symbolic manipulation and evaluation
5. Control – we have already seen "straight line execution, and if statement.

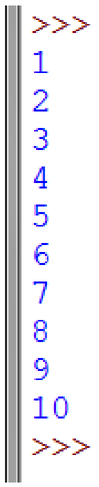
We have seen a simple example of control. Python simply executes one statement after another from the first statement of the program to the last i.e. straight line execution. In we have seen how Python lets us execute statements **conditionally**. So if a Boolean condition is met, a given block of statements is executed **once**, otherwise it is not.

But … what about the case that we want to execute the block multiple times, as long as the Boolean condition is True? We can do this by using a

while loop.

Example:

Write a program that prints the numbers 1 – 10, one per line.



```
>>>
1
2
3
4
5
6
7
8
9
10
>>>
```

Solution: Use a while loop.

```
i=1 # "initialize" variable i, give it its initial (first) value
while i<=10:
    print(i)
    i+=1 #increment i by 1
```

The **syntax** (form) of the while statement is:

while <Boolean expression> :
    one or more statements (called a code block)

The **semantics** (meaning or interpretation) is:

Upon encountering the while statement:

1. Evaluate the Boolean expression
2. Repeat the associated block as long as the Boolean expression is True
3. Exit the **while loop** as soon as the Boolean expression evaluates to False.

Definition: This process of repeating one or more statements is called **iteration**.

Problem:

Write a program that asks the user for a positive integer n. It then prints the numbers 1 through n, one per line.

Answer:

Problem:

Write a program that asks the user for a positive integer n. It then prints the numbers **n down to 1,** one per line.

Answer:

Problem:

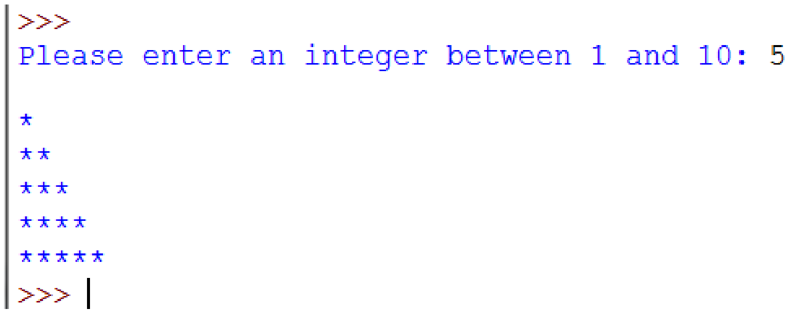
Write a program that asks the user for a positive integer n (>= 2). It then prints the even numbers in the range 1 to n, one per line.

Answer:

L3_even_upTo.py

Problem:

Write a program that asks the user for a positive integer n. Print a triangle of “stars”(= “*”) like in the example below.



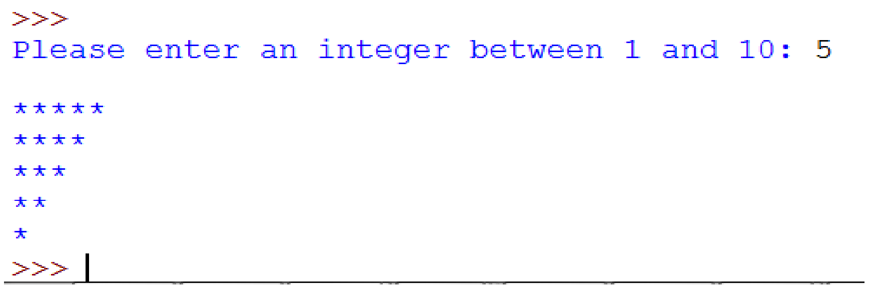
```
>>>
Please enter an integer between 1 and 10: 5

*
**
***
****
*****
>>> |
```

Answer:

L3_triangle_star.py

Problem:

Now do this one.



```
>>>
Please enter an integer between 1 and 10: 5

*****
****
***
**
*
>>> |
```

L3_triangle_star2.py

Problem:

Write a program that asks the user for a positive integer n. Calculate and print the **sum** of the integers 1 through n.

Answer:

L3_sum_upTo.py

Problem:

Write a program that asks the user for a positive integer n. Calculate and print the **product** of the integers 1 through n.

Answer:

Problem:

Write a program that asks the user for a positive integer n. Calculate and print the **sum of the odd integers** in the range 1 through n.

Answer:

L3_range_sumOdd.py

Problem:

Write a program to **request and validate a password** from your user. A valid password will be any two digit integer, **both digits of which are even**.

Give the user **three** chances to enter a correct password.

- At each incorrect attempt, print "Invalid password. Try again"
- If the password entered is correct print "Correct! You may access the system." Exit the program.
- If the password entered is incorrect print "Too many invalid attempts. Please try again later."

Answer:

L3_password.py

Problem:

Write a program that asks the user for a positive integer n where the right-most digit is not a zero. Print out the digits of n from right to left – one next to the other.

Answer:

Problem:

Write a program that asks the user for a positive integer n where the right-most digit is not a zero. **Construct and output the integer whose digits are the reverse of those in n.**

For example, if n has the value 123, then you need to construct the integer 321. **Note** you are not just printing the digits of n in reverse order; you are actually constructing the new integer.

Answer:

1. What is the algorithm (method)?

2. Write the code.

L3_reverse2.py

Problem:

Recall the definition of a prime number:

An integer greater than one is called a prime number if its only positive divisors (factors) are one and itself.

Write a program that inputs a positive integer n and determines if n is prime or composite (i.e. not a prime).

Answer:

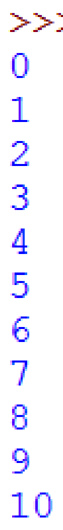
L3_prime.py

We now look at the second loop structure in Python, the

for loop

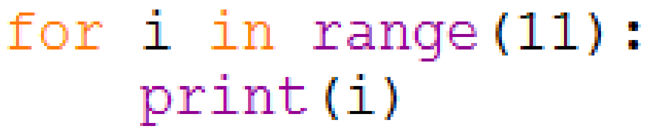
Example:

Write a program that prints the numbers 0 – 10, one per line.



```
>>>
0
1
2
3
4
5
6
7
8
9
10
```

Answer:



```
for i in range(11):
    print(i)
```

This is the simplest usage of the for statement. It lets you iterate over a sequence of numbers. In the example above, the sequence starts implicitly at 0 and goes up to, but not including 11.

The **syntax** (form) of this form of the for statement is:

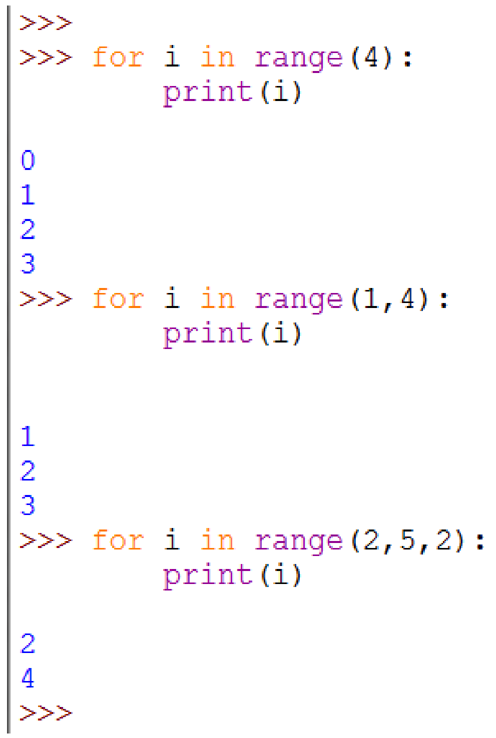
for <some variable> in range(…) :
    one or more statements (called a code block)

The **semantics** (meaning or interpretation) is:

- The variable after the "for" takes on each value in the "range" successively.
- That value may be used in the block that is the body of the for statement.

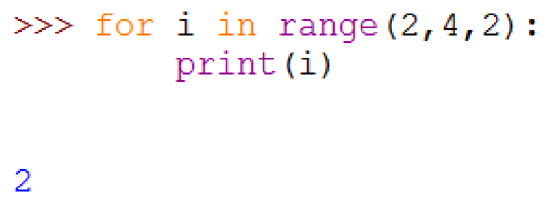
Question: What about **the range function**?

Here are some examples:



```
>>> 
>>> for i in range(4):
	print(i)

	
0
1
2
3
>>> for i in range(1,4):
	print(i)

	

1
2
3
>>> for i in range(2,5,2):
	print(i)

	
2
4
>>> 
```

and



```
>>> for i in range(2,4,2):
	print(i)

	
2
```

range(…) has the following **syntax**:

range([start value,] end value [,step])

and the following **semantics**:

1. if start value > end value:

- Generate a sequence of values starting with the start value
- generate each subsequent value by adding "step" to the previous one
- the last value in the sequence is the **<u>largest</u>** number **<u>less than</u>** end value.

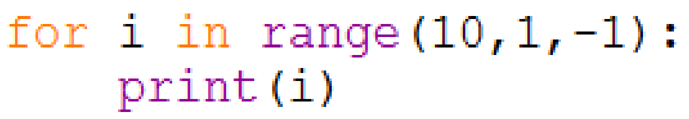
Note: The default value of "step" is 1.

2. if start value < end value:

- Generate a sequence of values starting with the start value
- generate each subsequent value by adding “step” to the previous one
- the last value in the sequence is the **<u>smallest</u>** number **<u>greater than</u>** end value.

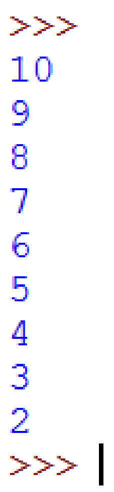
Note: In this case the value of “step” should be negative.

Example: The following code:



```python
for i in range(10,1,-1):
    print(i)
```

prints:



```
>>>
10
9
8
7
6
5
4
3
2
>>> |
```

Use a for loop to solve the next set of problems.

Problem:

Write a program that asks the user for a positive integer n. It then prints the numbers 1 through n, one per line.

Answer:

Problem:

Write a program that asks the user for a positive integer n. Calculate and print the **sum** of the integers 1 through n.

Answer:

Problem:

Write a program that asks the user for a positive integer n. Calculate and print the **product** of the integers 1 through n.

Answer:

Problem:

Write a program that asks the user for a positive integer n. Calculate and print the **sum of the odd integers** in the range 1 through n.

Answer:

L3_range_sumOdd.py

Problem:

Generate and print all numbers between 1 and 1000 such that the sum of the digits equals 20.

Answer:

L3_sumDigits.py

Problem:

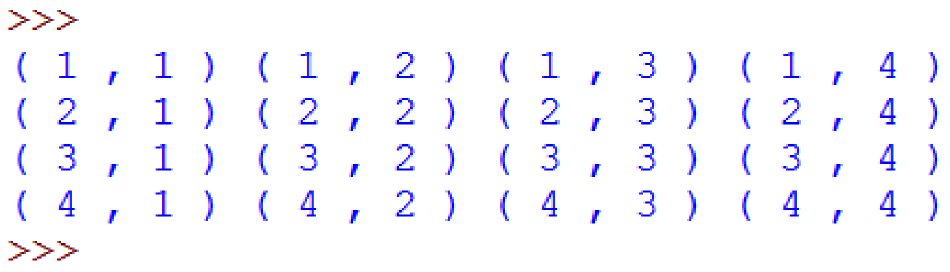
Generate and print all prime numbers between 1 and 100.

Answer:

L3_primes_between.py

Problem:

Write a program using for loops to print the following:



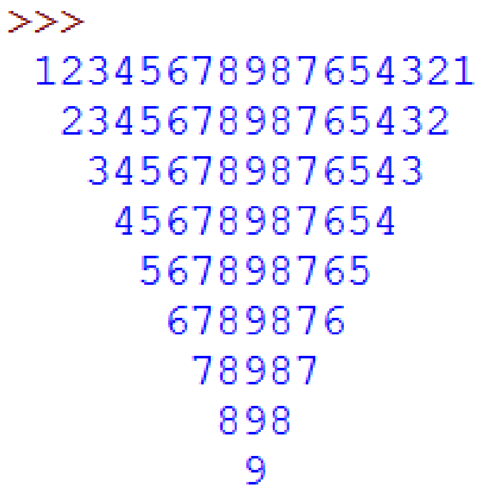
```
>>>
( 1 , 1 ) ( 1 , 2 ) ( 1 , 3 ) ( 1 , 4 )
( 2 , 1 ) ( 2 , 2 ) ( 2 , 3 ) ( 2 , 4 )
( 3 , 1 ) ( 3 , 2 ) ( 3 , 3 ) ( 3 , 4 )
( 4 , 1 ) ( 4 , 2 ) ( 4 , 3 ) ( 4 , 4 )
>>>
```

Answer:

L3_pairs.py

Problem:

Write a program, using for loops, to generate the following output.



```
>>>
 12345678987654321
  234567898765432
   3456789876543
    45678987654
     567898765
      6789876
       78987
        898
         9
```

Answer:

L3_triangle_upsideDown.py

How do we control statement execution inside a loop (either while or for)? Two new ways.

Until now:

while loop:

We keep executing the body of the loop as long as the Boolean expression in the loop head is true. We exit only when it becomes false.

for loop:

We keep executing the body of the loop as long as the sequence of values in the range() function has not been exhausted. We exit only when there are no more values generated in conjunction with the "argument list".

However …. Python has two other statements that allow us to control what happens inside a loop:

1. **continue**
2. **break**

1. continue: When Python encounters a **continue statement** inside a loop, the interpreter proceeds directly to the top of the loop, skipping all the statements in the after the continue.

For example this code

```
for i in range(1,5):
    for j in range(1,5):
        if (i+j)%2==0:
            continue
        print('(',i,',',j,')',end='  ')
    print()
```

produces this output:

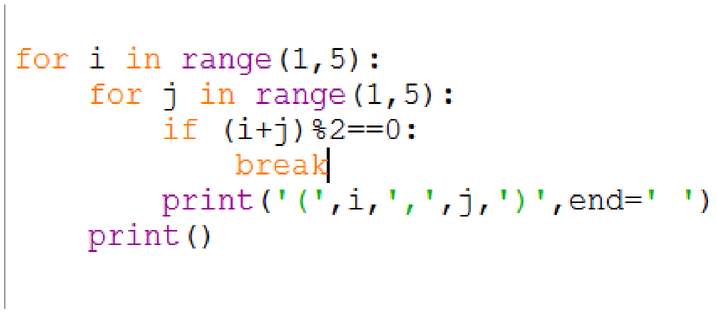
```
>>>
( 1 , 2 ) ( 1 , 4 )
( 2 , 1 ) ( 2 , 3 )
( 3 , 2 ) ( 3 , 4 )
( 4 , 1 ) ( 4 , 3 )
>>>
```

Question: What would be printed if the if and continue are left out?

2. break: When Python encounters a **break statement** inside a loop, the interpreter causes the loop to terminate and execution of the program continues with the first statement after the loop.

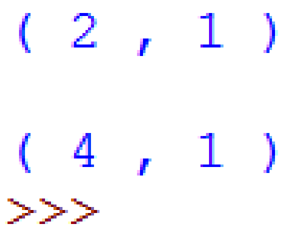
For example this code

For example this code



```python
for i in range(1,5):
    for j in range(1,5):
        if (i+j)%2==0:
            break
        print('(',i,',',j,')',end=' ')
    print()
```

produces this output:



```
( 2 , 1 )

( 4 , 1 )
>>>
```

Question: explain this output in detail, including the blank line between the tuples printed.

Here is an attempt to re-write the “continue” example with the for loop except using a while loop:

example

```
i=1
j=1
while i <5:
    while j <5:
        if (i+j)%2==0:
            continue
        print('(',i,',',j,')',end=' ')
        j+=1
    print()
    i+=1
```

Question: What will be printed? Why?

If instead of a **continue** statement in the code above, we put a **break**, like this:

```
i=1
j=1
while i <5:
    while j <5:
        if (i+j)%2==0:
            break
        print('(',i,',',j,')',end=' ')
        j+=1
    print()
    i+=1
```

we get

```
( 2 , 1 )
( 3 , 2 )
( 4 , 3 )
>>>
```

Why?