2019

Python loops

Natalia Novak

Bronx Community College, City University of New York

How does access to this work benefit you? Let us know!

Follow this and additional works at: https://academicworks.cuny.edu/bx_oers

Part of the Other Computer Sciences Commons

Recommended Citation

https://academicworks.cuny.edu/bx_oers/36

This Lecture or Presentation is brought to you for free and open access by the Bronx Community College at CUNY Academic Works. It has been accepted for inclusion in Open Educational Resources by an authorized administrator of CUNY Academic Works. For more information, please contact AcademicWorks@cuny.edu.
Loops

- While Loops
- For Loops
- Nested loops
- Break and continue
Loops

What is a loop?

Loops are all about repeating the same behavior for some number of times or until some condition is met.

**Example:** recall our Candy Quest

```python
repeat until "red candy"
    if "bottle cap":
        jump
    else:
        walk
```
While loops

Two types of loop in Python

In Python we have two types of loops:

while loop: executes a block of code as long as the loop's expression is True.

```
while <condition>:
    # loop body
    # loop body
    # statements to execute when condition becomes False
```
Two types of loop in Python

In Python we have two types of loops:

**for loop**: a counted loop, executes a block of code \( n \) times

```python
for variable in container:
    # Loop body
    # Loop body
```

# Statements to execute after the for loop is complete
While loops

Let's start with **while loops**:
Let’s start with **while loops**:
Let's start with **while loops**:
While loops

Let’s start with **while loops**:

```
jump
jump
jump
jump
jump
```
Let’s start with **while loops**:
Let’s start with **while loops**:
While loops

Let’s start with *while loops*:
Let's start with `while` loops:

```plaintext
while the star is not reached:
    jump
```

😊
Let's start with while loops:

while the star is not reached:
    jump

😊
Let’s start with **while loops**:

```plaintext
while the star is not reached:
    jump
```
While loops

Let’s start with **while loops**:

```plaintext
while the star is not reached:
  jump
```

While loops
While loops

Let’s start with **while loops**:

```plaintext
while the star is not reached:
  jump
```

Let’s start with **while loops**:

```text
while the star is not reached:
  jump
```
While loops

Let’s start with while loops:

while the star is not reached:
  jump
While loops

Let’s start with while loops :

while the star is not reached:
    jump
While loops

Let's start with **while loops**: 

while the star is not reached:

jump
Let’s start with **while loops**:

```python
while the star is not reached:
    jump
```
While loops

Let’s start with **while loops**: 

```plaintext
while the star is not reached:
  jump
```

While loops
Let’s start with **while loops**: while the star is not reached:

- jump
Let’s start with **while loops**:

while the star is not reached:

*jump*
Let's talk about while loops.

```python
x = int(input("Enter a positive integer:

while x > 0:
    print("***")
    x = x - 3
print("the end")
```
Let's start with **while loops**.

```python
x = int(input("Enter a positive integer:"))
while x > 0:
    print("***")
    x = x - 3
print("the end")
```
Let's start with while loops.

```
x = int(input("Enter a positive integer:"))

while x > 0:
    print("***")
    x = x - 3
print("the end")
```

Enter a positive integer: 7
Let’s start with while loops.

```python
x = int(input("Enter a positive integer:"))

while x > 0:
    print("***")
    x = x - 3
    print("the end")
```

Enter a positive integer: 7

x = 7
Let's start with while loops.

```python
x = int(input("Enter a positive integer:"))

while x > 0:
    print("***")
    x = x - 3

print("the end")
```

Enter a positive integer: 7

***
x=7
Let's start with `while` loops.

```python
x = int(input("Enter a positive integer:"))

while x > 0:
    print("***")
    x = x - 3
print("the end")
```

Enter a positive integer: 7
***

x=4
While loops

Let’s start with **while loops**.

```python
x = int(input("Enter a positive integer:

while x > 0:
    print("***")
    x = x - 3
print("the end")
```

Enter a positive integer:7
***

x=4
Let's start with `while` loops.

```python
x = int(input("Enter a positive integer:"))

while x > 0:
    print("***")
    x = x - 3
print("the end")
```

Enter a positive integer: 7

```
***
***
x=4
```
Let's start with while loops.

```python
x = int(input("Enter a positive integer:"))

while x > 0:
    print("***")
    x = x - 3
    print("the end")
```

Enter a positive integer: 7

***
 ***

x=1
Let's start with while loops.

```python
x = int(input("Enter a positive integer:"))

while x > 0:
    print("***")
    x = x - 3
    print("the end")
```

Enter a positive integer: 7
***
***
x = 1
Let’s start with while loops.

```python
x = int(input("Enter a positive integer:"))

while x > 0:
    print("***")
    x = x - 3
print("the end")
```

Enter a positive integer: 7

```python
***
***
***
```

x = 1
Let’s start with `while` loops.

```python
x = int(input("Enter a positive integer:"))) 
while x > 0:
    print("***")
    x = x - 3
print("the end")
```

Enter a positive integer:7
***
***
***

x=-2
Let’s start with while loops.

```python
x = int(input(“Enter a positive integer:”))
while x > 0:
    print(“***”)
    x = x - 3
print(“the end”)
```

Enter a positive integer: 7
***
***
***

x = -2
Let's start with while loops.

```python
x = int(input("Enter a positive integer:"))

while x > 0:
    print("***")
    x = x - 3
print("the end")
```

Enter a positive integer: 7
***
***
***
the end

x = -2
Let’s start with while loops.

```python
x = int(input("Enter a positive integer:"))

while x > 0:
    print("***")
    x = x - 3
    print("the end")
```

Each execution of the loop body is called an iteration, and looping is also called iterating.
While loops: common errors

An *infinite loop* is a loop that will always execute because the loop's expression is always **True**.

A common error is to accidentally create an infinite loop by assuming equality will be reached.
While loops

While loops: common errors

An *infinite loop* is a loop that will always execute because the loop's expression is always *True*.

A common error is to accidentally create an infinite loop by assuming equality will be reached.

```python
x = int(input("Enter a positive integer:"))

while x = 0:
    print("***")
    x = x - 3
print("the end")
```
While loops

While loops: common errors

An infinite loop is a loop that will always execute because the loop's expression is always True.

A common error is to accidentally create an infinite loop by assuming equality will be reached.

```python
x = int(input(“Enter a positive integer:”))
while x = 0:
    print(“***”)
    x = x - 3
print(“the end”)
```

x = 7, 4, -1, -4, ...
While Loops

Let's take a look at the programs with various loops:

whileLoopExamples1.py
whileLoopExamples2.py
whileLoopExamples3.py
whileLoopExamples4.py
While Loops

In-class work

See the handout, items 1-3
For loops

Another type of loop in Python

For loop: a counted loop, executes a block of code \( n \) times; iterates over the elements in a container

```
for variable in container:
    # Loop body
    # Loop body

# Statements to execute after the for loop
# is complete
```
Another type of loop in Python

**for loop**: a counted loop, executes a block of code *n* times; iterates over the elements in a container

```python
for item in [10, 20, 35, 43, 56, 90]:
    print(2*item, end="", )
print("finished!")
```
**For loops**

Another type of loop in Python

**for loop**: a counted loop, executes a block of code \( n \) times;
Iterates over the elements in a container

```python
for item in [10, 20, 35, 43, 56, 90]:
    print(2*item, end="", )
print("finished!")
```
For loops

Another type of loop in Python

for loop: a counted loop, executes a block of code \( n \) times; Iterates over the elements in a container

```python
for item in [10,20,35,43,56,90]:
    print(2*item, end="", )

print("finished!")
```

20,
Another type of loop in Python

**for loop**: a counted loop, executes a block of code \( n \) times; iterates over the elements in a container

```python
for item in [10, 20, 35, 43, 56, 90]:
    print(2*item, end="", )

print("finished!")
```

20,
For loops

Another type of loop in Python

for loop: a counted loop, executes a block of code \( n \) times; iterates over the elements in a container

```python
for item in [10,20,35,43,56,90]:
    print(2*item, end="", )

print("finished!")
```

20, 40,
**For loops**

Another type of loop in Python

*for loop*: a counted loop, executes a block of code \( n \) times; iterates over the elements in a container

```python
for item in [10,20,35,43,56,90]:
    print(2*item, end="", "")
print("finished!")
```

20, 40,
For loops

Another type of loop in Python

for loop: a counted loop, executes a block of code n times; iterates over the elements in a container

for item in [10, 20, 35, 43, 56, 90]:
    print(2*item, end="", )
print("finished!")

20, 40, 70,
Another type of loop in Python

for loop: a counted loop, executes a block of code \( n \) times; iterates over the elements in a container

```python
for item in [10, 20, 35, 43, 56, 90]:
    print(2*item, end="", )
print("finished!")
```

20, 40, 70,
For loops

Another type of loop in Python

for loop: a counted loop, executes a block of code \( n \) times;
Iterates over the elements in a container

```python
for item in [10, 20, 35, 43, 56, 90]:
    print(2*item, end="", )
print("finished!")
```

20, 40, 70, 86,
For loops

Another type of loop in Python

for loop: a counted loop, executes a block of code \( n \) times; iterates over the elements in a container

```python
for item in [10, 20, 35, 43, 56, 90]:
    print(2*item, end="", )
print("finished!")

20, 40, 70, 86,
```
For loops

Another type of loop in Python

for loop: a counted loop, executes a block of code \( n \) times;
Iterates over the elements in a container

for item in [10, 20, 35, 43, 56, 90]:
    print(2*item, end="", )
print("finished!")

20, 40, 70, 86, 112,
Another type of loop in Python

A for loop is a counted loop, which executes a block of code $n$ times. It iterates over the elements in a container. For example:

```python
for item in [10, 20, 35, 43, 56, 90]:
    print(2*item, end="", )
print("finished!")
```

This prints:

```
20, 40, 70, 86, 112,
```
For loops

Another type of loop in Python

*for loop*: a counted loop, executes a block of code \( n \) times; iterates over the elements in a container

```python
for item in [10, 20, 35, 43, 56, 90]:
    print(2*item, end="", "")
print("finished!")
```

20, 40, 70, 86, 112, 180,
For loops

Another type of loop in Python

for loop: a counted loop, executes a block of code \( n \) times; iterates over the elements in a container

```python
for item in [10, 20, 35, 43, 56, 90]:
    print(2*item, end="", "")
print("finished!")

20, 40, 70, 86, 112, 180,
```
Another type of loop in Python

**for loop**: a counted loop, executes a block of code \( n \) times; iterates over the elements in a container

```python
for item in [10, 20, 35, 43, 56, 90]:
    print(2*item, end="", )
print("finished!")
```

20, 40, 70, 86, 112, 180, finished!
Another type of loop in Python

Example:
```

for k in myD:
    print("%d corresponds to %s" % (k,myD[k]))

print("finished!")
```
For loops

Another type of loop in Python

Example:

for k in myD:
    print("%d corresponds to %s" % (k, myD[k]))
print("finished!")

172 corresponds to Friday
823 corresponds to Tuesday
564 corresponds to Monday
923 corresponds to Saturday
...
For loops

Example: Assume that we are given a list of decimal values in a Python list, named data. And we are asked to write a program that finds the average of all the values in the list data.
**For loops**

**Example:** Assume that we are given a list of decimal values in a Python list, named `data`. And we are asked to write a program that finds the average of all the values in the list `data`.

Here is an order of operations to do:

- add the values from list `data` one by one to a variable `s`,
- count the number of values in the list `data`,
- get the average by dividing `s` by `count`. 
For loops

Example: Assume that we are given a list of decimal values in a Python list, named data. And we are asked to write a program that finds the average of all the values in the list data.

Here is a program for that:

```python
s = 0
count = 0
for val in data:
    s += val
    count += 1
average = s / count
print("The average of the values in the list", data,"is", average)
```
**Example:** Assume that we are given a list of decimal values in a Python list, named `data`. And we are asked to write a program that finds the average of all the values in the list `data`.

Here is a program for that:

```
s = 0
count = 0
for val in data:
    s += val
    count += 1
average = s / count
print("The average of the values in the list",
data,"is", average)
```

How can we simplify the program? **hint:** I don’t need to count the number of values in the list `data`.
**Example**: Assume that we are given a list of decimal values in a Python list, named `data`. And we are asked to write a program that finds the **average** of all the values in the list `data`.

Here is a program for that:

```python
s = 0
count = 0
for val in data:
    s += val
    count += 1
average = s / count
print("The average of the values in the list",
data,"is", average)
```

How can we simplify the program?

*hint*: I don't need to count the number of values in the list `data`. Use `len`.
For loops

**Example:** Assume that we are given a list of decimal values in a Python list, named `data`. And we are asked to write a program that finds the average of all the values in the list `data`.

Here is a program for that:

```python
s = 0
for val in data:
    s += val

average = s / len(data)

print("The average of the values in the list", data, "is", average)
```
For loops

**Example**: Assume that we are given a list of decimal values in a Python list, named `data`. And we are asked to write a program that finds the **average** of all the values in the list `data`.

**In-class work:**

**Exercise 1**: modify the program we wrote to find the **average** of all negative numbers in the list `data`. Send the code to my e-mail.
**Example:** Now let’s consider a small database with names associated with the e-mail addresses:

```javascript
contactInfo = {
    "Mark Huggard": "Mhuggard@org.com",
    "Alice True": "ATrue@org2.com",
    "Hunter O'Brien": "HOBrien@org.com",
    "Jane Cole": "JaneCole@org6.com",
    "Frank Dove": "FrankDove@org7.com"
}
```
**Example:** Now let’s consider a small database with names associated with the e-mail addresses:

```python
contactInfo = {
    "Mark Huggard" : "Mhuggard@org.com",
    "Alice True" : "ATrue@org2.com",
    "Hunter O’Brien" : "HOBrien@org.com",
    "Jane Cole" : "JaneCole@org6.com",
    "Frank Dove" : "FrankDove@org7.com"
}
```

Now, let’s write the code fragment to:

1) extract all e-mail addresses from `contactInfo` and store them in a Python list, named `emails`, then
2) display alphabetically sorted list of e-mails.
For loops

Example: Now let’s consider a small database with names associated with the e-mail addresses:

```python
contactInfo = {
    "Mark Huggard": "Mhuggard@org.com",
    "Alice True": "ATrue@org2.com",
    "Hunter O'Brien": "HOBrien@org.com",
    "Jane Cole": "JaneCole@org6.com",
    "Frank Dove": "FrankDove@org7.com"
}

emails = []
for k in contactInfo:  # extract all e-mail addresses
    emails.append(contactInfo[k])
emails.sort()  # sort alphabetically
print(emails)  # display
```
### Counting using the `range()` function

<table>
<thead>
<tr>
<th>Range</th>
<th>Generated sequence</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>range(5)</code></td>
<td>0 1 2 3 4</td>
<td>every integer from 0 to 4</td>
</tr>
<tr>
<td><code>range(0, 5)</code></td>
<td>0 1 2 3 4</td>
<td>every integer from 0 to 4</td>
</tr>
<tr>
<td><code>range(3, 7)</code></td>
<td>3 4 5 6</td>
<td>every integer from 3 to 6</td>
</tr>
<tr>
<td><code>range(10, 13)</code></td>
<td>10 11 12</td>
<td>every integer from 10 to 12</td>
</tr>
<tr>
<td><code>range(0, 5, 1)</code></td>
<td>0 1 2 3 4</td>
<td>every 1 integers from 0 to 4</td>
</tr>
<tr>
<td><code>range(0, 5, 2)</code></td>
<td>0 2 4</td>
<td>every 2 integers from 0 to 4</td>
</tr>
<tr>
<td><code>range(5, 0, -1)</code></td>
<td>5 4 3 2 1</td>
<td>every 1 integer from 5 to 1</td>
</tr>
<tr>
<td><code>range(5, 0, -2)</code></td>
<td>5 3 1</td>
<td>every 2 integers from 5 to 1</td>
</tr>
</tbody>
</table>
**Example:** Prices of various items change with time. Let’s write the program that will allow us to calculate the price of the smartphone few years later, if we are given this year’s price, and each year’s inflation rate in percent.

```python
price = float(input("Enter phone's price:"))
n = int(input("Enter the number of years:"))
rate = float(input("Enter the inflation rate:"))
rate_decimal = rate/100
```
Counting using the `range()` function

**Example**: Prices of various items change with time. Let’s write the program that will allow us to calculate the price of the smartphone few years later, if we are given this year’s price, and each year’s inflation rate in percent.

```
price = float(input("Enter phone's price:"))
n = int(input("Enter the number of years:"))
rate = float(input("Enter the inflation rate:"))
rate_decimal = rate/100

newPrice = price
for i in range(0,n):
    newPrice = newPrice + newPrice*rate_decimal

print("The phone's price will be $%f" % newPrice)
```
Counting using the `range()` function

In-class activity

Do the exercises 4-6 from the handout
While vs. for loops

Consider the following scenarios:

(a) iterate until the user enters a letter ‘p’ or a letter ‘q’

(b) iterate 50 times

(c) iterate until x is less than 19

(d) find the product of all values in a list

What looping mechanism to use? while or for?
While vs. for loops

Consider the following scenarios:

(a) iterate until the user enters a letter ‘p’ or a letter ‘q’  
   while loop

(b) iterate 50 times  
   for loop

(c) iterate until x is less than 19  
   while loop

(d) find the product of all values in a list  
   for loop

What looping mechanism to use? while or for?
While vs. for loops

Here is a guideline:

• The number of iterations is computable before entering the loop → use the **for loop**
  **examples:** counting down from \(X\) to 0, printing a string \(N\) times, etc.

• Accessing the elements of a container → use the **for loop**
  **examples:** when adding 1 to every element in a list, or printing the key of every entry in a dict, etc.

• The number of iterations is not computable before entering the loop → use the **while**
  **example:** when iterating until a user enters a particular character.
While vs. for loops

Example: Given a string `myString`, let’s write a program to count the number of occurrences of letter ‘a’ in it, without using built-in function `count()`.
Example: Given a string `myString`, let’s write a program to count the number of occurrences of letter ‘a’ in it, without using built-in function `count()`.

1) What looping mechanism to use (`while` or `for`)?
While vs. for loops

**Example:** Given a string `myString`, let’s write a program to count the number of occurrences of letter ‘a’ in it, without using built-in function `count()`.

1) What looping mechanism to use (while or for)? **for loop**
While vs. for loops

Example: Given a string `myString`, let’s write a program to count the number of occurrences of letter ‘a’ in it, without using built-in function `count()`.

1) What looping mechanism to use (while or for)? **for loop**

2) let’s write the code:

```python
counter = 0
for ch in myString:
    if ch == 'a': counter += 1
# at the end of loop’s executions, variable
# counter will have the number of occurrences
# of letter ‘a’ in myString
```
While vs. for loops

In-class activity

Do the exercises 4-5 from the handout
Nested loops

A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10,20,30,40]:
    for y in range(1,10):
        print(x+y,end = " ")
print()
```
Nested loops

A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10, 20, 30, 40]:
    for y in range(1, 10):
        print(x+y, end = " ")
        print()
```

- **x →** 10, 20, 30, 40
- **y →** 1, 2, 3, 4, 5, 6, 7, 8, 9
A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10,20,30,40]:
    x → 10
    for y in range(1,10):
        y → 1
        print(x+y,end = " ")
    print()
```
Nested loops

A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10, 20, 30, 40]:
    for y in range(1, 10):
        print(x+y,end = " ")
print()
```

11

11
Nested loops

A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10, 20, 30, 40]:
    for y in range(1, 10):
        print(x+y, end = " ")
        y → 2
print()
```

11 12
Nested loops

A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10,20,30,40]:
    for y in range(1,10):
        print(x+y,end = " ")
print()
```

```
11 12 13 14 15 16 17 18 19
```
Nested loops

A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10,20,30,40]:
    for y in range(1,10):
        print(x+y,end = " ")
    print()
```

```
11 12 13 14 15 16 17 18 19
```
Nested loops

A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10,20,30,40]:
    for y in range(1,10):
        print(x+y,end = " ")
    print()
```

11 12 13 14 15 16 17 18 19
Nested loops

A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10,20,30,40]:
    for y in range(1,10):
        print(x+y,end = " ")
print()
```

```
11 12 13 14 15 16 17 18 19
21
```
A loop that appears as part of the body of another loop is called a **nested loop**.

The nested loops are commonly referred to as the **outer loop** and **inner loop**.

```python
for x in [10, 20, 30, 40]:
    for y in range(1, 10):
        print(x+y, end = " ")
print()
```

```
11 12 13 14 15 16 17 18 19
21
```
A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
def print(x, y, end):
    print(x+y, end=end)

for x in [10, 20, 30, 40]:
    for y in range(1, 10):
        print(x+y, end=" ")
print()```

```
11 12 13 14 15 16 17 18 19
21 22
```
Nested loops

A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10, 20, 30, 40]:
    x → 20
    for y in range(1, 10):
        y → 9
        print(x+y,end = " ")
    print()
```

11 12 13 14 15 16 17 18 19
21 22 23 24 25 26 27 28 29
Nested loops

A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10, 20, 30, 40]:
    for y in range(1, 10):
        print(x+y, end = " ")
    print()
```

x → 30
y → 1, 2, 3, 4, 5, 6, 7, 8, 9

11 12 13 14 15 16 17 18 19
21 22 23 24 25 26 27 28 29
Nested loops

A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10, 20, 30, 40]:
    for y in range(1, 10):
        print(x + y, end = " ")
    print()
```

```
11 12 13 14 15 16 17 18 19
21 22 23 24 25 26 27 28 29
31 32 33 34 35 36 37 38 39
```
Nested loops

A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10, 20, 30, 40]:
    for y in range(1, 10):
        print(x + y, end = " ")
    print()
```

```plaintext
11 12 13 14 15 16 17 18 19  
21 22 23 24 25 26 27 28 29  
31 32 33 34 35 36 37 38 39  
41 42 43 44 45 46 47 48 49  
```

x → 30
y → 1, 2, 3, 4, 5, 6, 7, 8, 9
A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```
for x in [10, 20, 30, 40]:
    for y in range(1, 10):
        print(x + y, end = " ")
    print()
```

```
x → 10, 20, 30, 40
y → 1, 2, 3, 4, 5, 6, 7, 8, 9
```

```
11 12 13 14 15 16 17 18 19
21 22 23 24 25 26 27 28 29
31 32 33 34 35 36 37 38 39
41 42 43 44 45 46 47 48 49
```
A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```python
for x in [10, 20, 30, 40]:
    for y in range(1, 10):
        print(x + y, end = " ")
        print()
```

How many entries are in the table on the left?
A loop that appears as part of the body of another loop is called a nested loop.

The nested loops are commonly referred to as the outer loop and inner loop.

```
for x in [10,20,30,40]:
    for y in range(1,10):
        print(x+y,end = " ")
    print()
```

How many entries are in the table on the left?

```
11 12 13 14 15 16 17 18 19
21 22 23 24 25 26 27 28 29
31 32 33 34 35 36 37 38 39
41 42 43 44 45 46 47 48 49
```

4 \times 9 = 36
Nested loops

**Example:** two-letter domain names

```python
def print_two_letters():
    letter1 = 'a'
    letter2 = 'a'
    while letter1 <= 'z':  # Outer loop
        letter2 = 'a'
        while letter2 <= 'z':  # Inner loop
            print('%s%s.info' % (letter1, letter2))
            letter2 = chr(ord(letter2) + 1)
        letter1 = chr(ord(letter1) + 1)

print_two_letters()
```

Nested loops

**Example:** two-letter domain names

```python
print('Two-letter domain names:
letter1 = 'a'
letter2 = 'a'
while letter1 <= 'z':  # outer loop
    letter2 = 'a'
    while letter2 <= 'z':  # inner loop
        print('%.s%.s.info
        letter2 = chr(ord(letter2) + 1)
    letter1 = chr(ord(letter1) + 1)
```
A break statement in a loop causes an immediate exit from the loop.

```python
while True:
    x = int(input("Enter an integer > 10:"))
    if x > 10:
        break
print("you made it!")
```
A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5, 20):
    if i%2 == 0:
        s *= (i//2)
    else:
        continue
print("s =", s)
```
A \texttt{continue} statement in a loop causes an immediate jump to the while or for loop header statement.

\begin{verbatim}
s = 1
for i in range(5, 20):
    if i%2 == 0:
        s *= (i//2)
    else:
        continue
print("s =", s)
\end{verbatim}
Break and continue

A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5, 20):
    if i % 2 == 0:
        s *= (i // 2)
    else:
        continue
print("s =", s)
```
Break and continue

A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5, 20):
    if i % 2 == 0:
        s *= (i // 2)
    else:
        continue
print("s =", s)
```

```
s = 1
i = 5
```

```
5 % 2 = 1
```
A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
def main():
    s = 1
    for i in range(5, 20):
        if i % 2 == 0:
            s *= (i//2)
        else:
            continue
    print("s =", s)

if __name__ == '__main__':
    main()
```

```python
s = 1
i = 5
5 % 2 = 1
```
Break and continue

A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5,20):
    if i%2 == 0:
        s *= (i//2)
    else:
        continue
print("s =", s)
```

```
s = 1
i = 5
5 % 2 = 1
```
A continue statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5, 20):
    if i%2 == 0:
        s *= (i//2)
    else:
        continue
print("s =", s)
```

Break and continue

A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5, 20):
    if i % 2 == 0:
        s *= (i // 2)
    else:
        continue
print("s =", s)
```
A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```
s = 1
for i in range(5, 20):
    if i%2 == 0:
        s *= (i//2)
    else:
        continue

print("s =", s)
```

```
s = 1
s = s*(6//2) = 1*3 = 3
i = 6
6 % 2 = 0
```
A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5, 20):
    if i % 2 == 0:
        s *= (i // 2)
    else:
        continue
print("s =", s)
```
A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5, 20):
    if i % 2 == 0:
        s *= (i // 2)
    else:
        continue
print("s =", s)
```

```
s = 3
i = 7
7 % 2 = 1
```
A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5, 20):
    if i%2 == 0:
        s *= (i//2)
    else:
        continue
print("s =", s)
```

```
s = 3  
i = 7
7 % 2 = 1
```
A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5,20):
    if i%2 == 0:
        s *= (i//2)
    else:
        continue
print("s =", s)
```

```python
s = 3
i = 8
```
A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5, 20):
    if i % 2 == 0:
        s *= (i//2)
    else:
        continue
print("s =", s)
```
A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5, 20):
    if i % 2 == 0:
        s *= (i // 2)
    else:
        continue
print("s =", s)
```

```
s = 3 * (8 // 2) = 12
i = 8
8 % 2 = 0
```

A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5,20):
    if i%2 == 0:
        s *= (i//2)
    else:
        continue
print("s =", s)
```

```
s = 12
i = 9
```
A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5, 20):
    if i % 2 == 0:
        s *= (i // 2)
    else:
        continue
print("s =", s)
```

s = 181440
i = 19
A `continue` statement in a loop causes an immediate jump to the while or for loop header statement.

```python
s = 1
for i in range(5, 20):
    if i % 2 == 0:
        s *= (i // 2)
    else:
        continue

print("s =", s)
```

```
s = 181440
i = 19
```

```
s = 181440
```
Nested loops, break and continue

In-class activity

Do the exercises 9-11 from the handout
This OER material was produced as a result of the CS04ALL CUNY OER project.

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 4.0 License.