**While you're here**

The while loop is similar to an ifstatement: it executes the code inside of it if some condition is true. The difference is that the while loop will continue to execute as long as the condition is true. In other words, instead of executing if something is true, it executes while that thing is true.

Line 6 decides when the loop will be executed. So, "as long as count is less than 5," the loop will continue to execute. Line 8 increases count by 1. This happens over and over until count equals 5.

# Condition

The condition is the expression that decides whether the loop is going to continue being executed or not. There are 5 steps to this program:

1. The loop\_condition variable is set to True
2. The while loop checks to see if loop\_condition is True. It is, so the loop is entered.
3. The print statement is executed.
4. The variable loop\_condition is set to False.
5. The while loop again checks to see if loop\_condition is True. It is not, so the loop is not executed a second time.

**While you're at it**

Inside a while loop, you can do anything you could do elsewhere, including arithmetic operations.

**Instructions**

**1.**

Create a while loop that prints out all the numbers from 1 to 10 squared (1, 4, 9, 16, ... , 100), each on their own line.

* Fill in the blank space so that our while loop goes from 1 to 10inclusive.
* Inside the loop, print the value of num squared. The syntax for squaring a number is num \*\* 2.
* Increment num.

|  |
| --- |
| * num = 1 * while num<=10: # Fill in the condition * # Print num squared * # Increment num (make sure to do this!) * print num\*\*2 * num +=1 |

# Simple errors

A common application of a while loop is to check user input to see if it is valid. For example, if you ask the user to enter y or n and they instead enter 7, then you should re-prompt them for input.

|  |
| --- |
| choice = raw\_input('Enjoying the course? (y/n)')  while choice != 'y' and choice != 'n': # Fill in the condition (before the colon)  choice = raw\_input("Sorry, I didn't catch that. Enter again: ") |

**Infinite loops**

An *infinite loop* is a loop that never exits. This can happen for a few reasons:

1. The loop condition cannot possibly be false (*e.g.* while 1 != 2)
2. The logic of the loop prevents the loop condition from becoming false.

Example:

count = 10 while count > 0: count += 1 # Instead of count -= 1

# Break

The break is a one-line statement that means "exit the current loop." An alternate way to make our counting loop exit and stop executing is with the break statement.

* First, create a while with a condition that is always true. The simplest way is shown.
* Using an if statement, you define the stopping condition. Inside the if, you write break, meaning "exit the loop."

The difference here is that this loop is guaranteed to run at least once.

|  |
| --- |
| count = 0  while True:  print count  count += 1  if count >= 10:  break |

# While / else

Something completely different about Python is the while/else construction. while/else is similar to if/else, but there is a difference: the else block will execute **anytime** the loop condition is evaluated to False. This means that it will execute if the loop is never entered or if the loop exits normally. If the loop exits as the result of a break, the else will not be executed.

In this example, the loop will break if a 5 is generated, and the else will not execute. Otherwise, after 3 numbers are generated, the loop condition will become false and the else will execute.

|  |
| --- |
| import random  print "Lucky Numbers! 3 numbers will be generated."  print "If one of them is a '5', you lose!"  count = 0  while count < 3:  num = random.randint(1, 6)  print num  if num == 5:  print "Sorry, you lose!"  break  count += 1  else:  print "You win!" |

|  |
| --- |
| Lucky Numbers! 3 numbers will be generated.  If one of them is a '5', you lose!  6  6  6  You win! |

# Your own while / else

Now you should be able to make a game similar to the one in the last exercise. The code from the last exercise is below:

count = 0 while count < 3: num = random.randint(1, 6) print num if num == 5: print "Sorry, you lose!" break count += 1 else: print "You win!"

In this exercise, allow the user to guess what the number is 3 times.

guess = int(raw\_input("Your guess: "))

Remember, raw\_input turns user input into a string, so we use int() to make it a number again.

**Instructions**

**1.**

Use a while loop to let the user keep guessing so long as guesses\_left is greater than zero.

Ask the user for their guess, just like the second example above.

If they guess correctly, print "You win!" and break.

Decrement guesses\_left by one.

Use an else: case after your whileloop to print "You lose.".

Hint

This game will have a very similar structure to the example, but instead of losing right before the break, the user should win.

The if should check if guess == random\_number. If it does, then it's the winning guess!

|  |
| --- |
| from random import randint  # Generates a number from 1 through 10 inclusive  random\_number = randint(1, 10)  guesses\_left = 3  # Start your game!  while guesses\_left > 0:  guess = int(raw\_input("Guess a number: "))  #print random\_number  if guess == random\_number:  print "You win!"  break  guesses\_left -= 1  else:  print "You lose." |

# For your health

An alternative way to loop is the for loop. The syntax is as shown in the code editor. This example means "for each number i in the range 0 - 9, print i".

|  |
| --- |
| print "Counting..."  for i in range(10):  print i |

# For your hobbies

This kind of loop is useful when you want to do something a certain number of times, such as append something to the end of a list.

**1.**

Create a for loop that prompts the user for a hobby 3 times.

Save the result of each prompt in a hobby variable

append each one to hobbies.

print hobbies after your for loop

Make sure to answer the prompts in the terminal when testing your code!

Hint

Your for loop should use range(3). You should use the raw\_input() function to get info from the user and hobbies.append(hobby) to add the hobby to the list.

|  |
| --- |
| hobbies = []  # Add your code below!  for i in range(3):  hobby = raw\_input("Inter: ")  hobbies.append(hobby)  print hobbies |
| Inter: dog  Inter: cat  Inter: lion  [u'dog', u'cat', u'lion'] |

|  |
| --- |
| hobbies = []  # Add your code below!  for i in range(3):  hobby = raw\_input("Inter: ")  hobbies.append(hobby)  print " ".join(hobbies) |
| Inter: dog  Inter: cat  Inter: lion  dog cat lion |

# For your strings

Using a for loop, you can print out each individual character in a string.

The example in the editor is almost plain English: "for each character c in thing, print c".

**Instructions**

**1.**

Add a second for loop so that each character in word is printed one at a

|  |
| --- |
| thing = "spam!"  for c in thing:  print c  word = "eggs!"  # Your code here!  for e in word:  print e |

**For your A**

String manipulation is useful in for loops if you want to modify some content in a string.

word = "Marble" for char in word: print char,

The example above iterates through each character in word and, in the end, prints out M a r b l e.

The , character after our print statement means that our next print statement keeps printing on the same line.

**Instructions**

**1.**

Let's filter out the letter "A" from our string.

* Do the following for each character in the phrase.
* If char is an "A" or char is an "a", print "X", instead of char. Make sure to include the trailing comma.
* Otherwise (else:), please print char, with the trailing comma.

Hint

You can use the same for syntax, for c in s, as before. Use an ifto compare c to 'a' and 'A'. Print an 'X' in that case, and use an else to print the character otherwise.

Include a comma after the character to be printed in order to ensure it's not printed on its own line, like so:

if c == "A" or c == "a": print "X",

|  |
| --- |
| phrase = "A bird in the hand..."  # Add your for loop  for char in phrase:  if char == "A" or char == "a":  print "X",  else:  print char,  #Don't delete this print statement!  print |
| X b i r d i n t h e h X n d . . . |

# For your lists

Perhaps the most useful (and most common) use of for loops is to go through a list.

On each iteration, the variable num will be the next value in the list. So, the first time through, it will be 7, the second time it will be 9, then 12, 54, 99, and then the loop will exit when there are no more values in the list.

**Instructions**

**1.**

Write a second for loop that goes through the numbers list and prints each element squared, each on its own line.

Hint

Use the \*\* operator for exponentiation. The rest of the loop should be very similar to the first one.

|  |
| --- |
| numbers = [7, 9, 12, 54, 99]  print "This list contains: "  for num in numbers:  print num  # Add your loop below!  for temp in numbers:  print temp\*\*2 |

**Looping over a dictionary**

You may be wondering how looping over a dictionary would work. Would you get the key or the value?

The short answer is: you get the key which you can use to get the value.

d = {'x': 9, 'y': 10, 'z': 20} for key in d: if d[key] == 10: print "This dictionary has the value 10!"

1. First, we create a dictionary with strings as the keys and numbers as the values.
2. Then, we iterate through the dictionary, each time storing the key in key.
3. Next, we check if that key's value is equal to 10.
4. If so, we print "This dictionary has the value 10!"

**Instructions**

**1.**

On line 5, print the key, followed by a space, followed by the value associated with that key.

Hint

An easy way to print in the requested format is to use the ,character, like so:

greeting = "Hello" name = "World" print greeting, name # prints "Hello World"

|  |
| --- |
| d = {'a': 'apple', 'b': 'berry', 'c': 'cherry'}  for key in d:  # Your code here!  print key, d[key] |

# Counting as you go

A weakness of using this for-each style of iteration is that you don't know the index of the thing you're looking at. Generally this isn't an issue, but at times it is useful to know how far into the list you are. Thankfully the built-in enumerate function helps with this.

enumerate works by supplying a corresponding index to each element in the list that you pass it. Each time you go through the loop, index will be one greater, and item will be the next item in the sequence. It's very similar to using a normal for loop with a list, except this gives us an easy way to count how many items we've seen so far.

**1.**

We don't want the user to see things listed from index 0, since this looks unnatural. Instead, the items should appear to start at index 1. Modify the print statement to reflect this behavior. See the Hint for help.

Hint

Instead of just printing index, print index + 1!

|  |
| --- |
| choices = ['pizza', 'pasta', 'salad', 'nachos']  print 'Your choices are:'  for index, item in enumerate(choices):  print index+1, item  print enumerate(choices) |
| Your choices are:  1 pizza  <enumerate object at 0x7f3bb49ef410>  2 pasta  <enumerate object at 0x7f3bb49ef410>  3 salad  <enumerate object at 0x7f3bb49ef410>  4 nachos  <enumerate object at 0x7f3bb49ef410> |

# Multiple lists

It's also common to need to iterate over two lists at once. This is where the built-in zipfunction comes in handy.

zip will create pairs of elements when passed two lists, and will stop at the end of the shorter list.

zip can handle three or more lists as well!

**Instructions**

**1.**

Compare each pair of elements and print the larger of the two.

Hint

a is an element from list\_a and b is an element of list\_b.

You have two options: Use an if/else statement to compare the two and print whichever is larger. Alternatively, you can use the maxfunction that you learned in unit 4.

|  |
| --- |
| list\_a = [3, 9, 17, 15, 19]  list\_b = [2, 4, 8, 10, 30, 40, 50, 60, 70, 80, 90]  for a, b in zip(list\_a, list\_b):  # Add your code here!  print max(a,b) |

# For / else

Just like with while, for loops may have an else associated with them.

In this case, the else statement is executed after the for, but only if the for ends normally—that is, not with a break. This code will break when it hits 'tomato', so the else block won't be executed.

**Instructions**

**1.**

Click Run to see how for and elsework together.

|  |
| --- |
| fruits = ['banana', 'apple', 'orange', 'tomato', 'pear', 'grape']  print 'You have...'  for f in fruits:  if f == 'tomato':  print 'A tomato is not a fruit!' # (It actually is.)  break  print 'A', f  else:  print 'A fine selection of fruits!' |
| You have...  A banana  A apple  A orange  A tomato is not a fruit! |

|  |
| --- |
| fruits = ['banana', 'apple', 'orange', 'pear', 'grape']  print 'You have...'  for f in fruits:  if f == 'tomato':  print 'A tomato is not a fruit!' # (It actually is.)  break  print 'A', f  else:  print 'A fine selection of fruits!' |
| You have...  A banana  A apple  A orange  A pear  A grape  A fine selection of fruits! |