## [Example 2: Given two strings, find all characters appearing in both.](http://www.pd4cs.org/example-2-given-two-strings-find-all-characters-appearing-in-both/)

We are given two strings and want to print the letters appearing in both strings. We use a for loop to iterate over one of the strings, string1, and compare every letter in string1 with letters in string2.

Compared to other languages, Python makes this easy and the following code is quite intuitive:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | # 02Loops example\_02 Strings.py  string1  = 'house'  string2 = 'our'    print "letters in both", string1, "and", string2, "are"    for letter in string1:  #Use each letter in string1 to test if it is in string2      if letter in string2:          print(letter)   #If a letter is also in string2, print it out |

## [Example 3: Print all the integers from 1 to 1000 that are multiples of both 6 and 10.](http://www.pd4cs.org/example-3-print-all-the-integers-from-1-to-1000-that-are-multiples-of-both-6-and-10/)

The program will again use the range function in a for loop. The first three integers printed will be 30, 60, and 90.

|  |  |
| --- | --- |
| 1  2  3  4 | # 02Loops example\_03 PrintIntegers.py  for number in range(1,1001):      if(number % 6 ==0 and number % 10 ==0):          print number |

Recall that % is the remainder operator (for example, 10%6 = 4).

## [Example 4: Better deal?](http://www.pd4cs.org/example-4-better-deal/)

Your rich uncle is giving his son $1,000,000 in cash.  Your father claims he loves you a lot, but he is only giving you pennies over a sequence of 30 days.  He will start giving you 1 penny on the first day and double the amount  every one of the next 29 days (i.e., 1 penny today, 2 pennies tomorrow, 4 pennies the next day, 8 pennies the next day, etc.).  Are you or your cousin getting the better deal?

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | # 02Loops example\_04 BetterDeal1.py  # who gets a better deal? Fixed amount or doubling pennies?    from \_\_future\_\_ import print\_function # use Python 3.0 printing    pennies = 1                #start with 1 penny on day 1  sumOfPennies = 0           #set the SumOfPennies to 0  days = 30  for day in range(1,days+1):    #loop for 30 days      sumOfPennies = sumOfPennies + pennies    #accumulate the sum of pennies      pennies = pennies \* 2  #pennies double daily    sumOfPennies = sumOfPennies / 100.0 #convert pennies to dollars    print ("The total amount you receive from your father is $" , sumOfPennies)    if sumOfPennies > 1000000:      print ("You are getting the better deal!")  else:      print("Your cousin is getting the better deal!") |

The output is:

The total amount you receive from your father is $ 10737418.23

You are getting the better deal!

The result of the program may be surprising to some students. It captures an important concept underlying many algorithms: if we take a number larger than one and double it repeatedly, the number grows exponentially. This means the number gets large very fast. This can be good (e.g., if it corresponds to money you get) or it can be bad (e.g., if it corresponds to the time needed to run an algorithm).

## [Example 5: Palindromes](http://www.pd4cs.org/example-5-palindromes/)

Palindrome is a word or phrase such as ‘deed’ or ‘level’, which is the same when you spell it backwards. The following program determines whether a word is palindrome.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | # 02Loops example\_05 Palindromes1.py  # determine whether a word input is a palimdrome    word = raw\_input("Enter a word: ")    revWord = ""                                #start reversed word as blank  x = len(word) - 1                           #find the length of the input word    for i in range(x,-1,-1):                    #start at end of word & move backwards      letter = word[i]                        #get the letter      revWord = revWord + letter              #add it to end of revWord    print "The original word was",word  print "The reversed word is",revWord    if (word == revWord):                  #check if the original word is same as reversed word          print word,"is a palindrome"  else:          print word,"is not a palindrome" |

For example, when we input “apple”, the output is:

Enter a word: apple

The original word was apple

The reversed word is elppa

apple is not a palindrome

When we input “level”, the output is:

Enter a word: level

The original word was level

The reversed word is level

level is a palindrome

In Python, there exists another way to reverse a string. Try the following

**>>> ‘foobar'[::-1]**

**‘raboof’**

Hence, we can write the following program:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | # 02Loops example\_05 Palindromes2.py  # determine whether a word input is a palimdrome    word = raw\_input("Enter a word: ")    revWord = word[::-1]               # generate the reversed string using slicing  print "The original word was",word  print "The reversed word is",revWord    if (word == revWord):          print word,"is a palindrome"  else:          print word,"is not a palindrome" |

In many languages, including Python, one task can be accomplished in a number of ways. Program Palindromes1 is the program that teaches students the logic needed to reverse a string in an iterative way.  Program Palindromes2 uses what Python calls “slicing” of strings. An introductory course would not teach slicing, but a few students are likely to find it. When giving students an assignment, it can be a good idea to state what libraries and features the students can and cannot use in their program

[**Example 6: Sum up the digits in an integer**](http://www.pd4cs.org/example-6-sum-up-the-digits-in-an-integer/)

Given a positive integer n, say 287591, we want to determine the sum of all its digits. The result for 287591 would be 2+8+7+5+9+1 = 32. To solve this problem, we need to determine the individual digits in integer n.

Our program works as follows:

* Use the remainder operator % to obtain the last digit of the integer and add the digit to the variable sum.
* Then, use the floor division operator //  to produce the integer without the last digit.
* Repeat these steps until there are no digits left.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | # 02Loops example\_06 SumDigits1.py  # add up the digits in an integer using a while loop    n = int(raw\_input("Please input an integer:"))    sum = 0  while n != 0:               #While there are still digits left in n, the loop continues      last\_digit = n % 10     #Obtain the last digit of n      sum = sum + last\_digit  #Add the last digit to sum      n = n // 10         #Remove the last digit of n with Floor Division operator    print(sum) |

Comments

* Video03NestedLoops2.mp4 uses this problem in a nested loop setting.
* The above program “destroys” integer n.  If for some reason we need n later, we would need to store it in another variable.
* If students have already seen for-loops in Python, ask them to try computing the sum of the digits with a for-loop. We will consider the for-loop version of this problem in Example 11.

## [Example 7: Guess a chosen letter in the alphabet](http://www.pd4cs.org/example-7-guess-a-chosen-letter-in-the-alphabet/)

We want to design a robot that can choose a lower case letter from the alphabet randomly. Then the user needs to guess which letter the robot has chosen. Like this:

Please guess my chosen letter: g

You are wrong! Try again!

Please guess my chosen letter: k

Great! You are right!

Solution with while loop. The event ending the loop is having guessed the right letter.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | #02Loops example\_07 GuessLetter.py    import random    alphabet = 'abcdefghijklmnopqrstuvwxyz'     #Initialize the alphabet  answer = random.choice(alphabet)        #Pick up a letter randomly  guess = raw\_input('Please guess my letter: ') #Ask the user to guess    while guess != answer:   #If the user's answer is not right, repeat the statements      print('You are wrong! Try again!')      guess = raw\_input('Please guess my letter: ')    print('Great! Your are right!') |

To help the user guess the right answer faster, we add a hint when the guess is wrong. The following is the revised code giving a hint:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | # 02Loops example\_07 GuessLetterWithHint.py  # with hint on whether guessed letter is too high ot too low    import random    alphabet = 'abcdefghijklmnopqrstuvwxyz'     #Initialize the alphabet  answer = random.choice(alphabet)        #Pick up a letter randomly  guess = raw\_input('Please guess my letter: ') #Ask the user to guess    while guess != answer:   #If the user's answer is not right, repeat the statements      if guess > answer:     #If the user抯 answer is not right, provide hints.          print('My letter is before your guess in the alphabet')      else:          print('My letter is after your guess in the alphabet')      guess = raw\_input('Please guess my letter: ')    print('Great! Your are right!') |

This code above is an example of what is known as binary search.

[**Example 8: The use of a sentinel value in while loops**](http://www.pd4cs.org/example-8-the-use-of-a-sentinel-value-in-while-loops/)

A sentinel value is a special value used to terminate a loop when reading data. In the following program, test scores are provided (via user input). Once the sentinel value of -1 is input, the loop terminates.  At that point, the average of the test scores will be printed.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | # 02Loops example\_08 SentinalValue.py    score = input("enter a test score, use -1 to stop ")    #enter the first value  total = 0  scoreCounter = 0    while score != -1:                      #loop until the sentinel value (-1) is entered          total = total + score          scoreCounter = scoreCounter + 1          score = input("enter a test score, use -1 to stop ")    print ("The average for the test is ",total/scoreCounter) |

This problem can be extended in a number of ways.

* Ask students to maintain the minimum and maximum score in the while loop.
* Ask students to store the values input in a list which can then be sorted and analyzed (e.g., determine the median score, 25% percentile, the score that appears most often).

Example: Print a table with three rows, each containing the numbers 1 2 3 4 5 6

The correct approach is to have an outer loop be responsible for printing the three rows. An inner loop is responsible for printing the six elements in each row.

|  |  |
| --- | --- |
| 1  2  3 | for i in range (1,4):       # controls the number of rows      for j in range (1,7):   #controls the number of columns          print(j) |

1 2 3 4 5 6

1 2 3 4 5 6

1 2 3 4 5 6

Swapping the order of outer and inner loop produces:

|  |  |
| --- | --- |
| 1  2  3 | for i in range (1,7):      for j in range (1,4):          print(j) |

1 2 3

1 2 3

1 2 3

1 2 3

1 2 3

1 2 3

## [Example 9: Converting a for-loop into a while loop](http://www.pd4cs.org/example-9-converting-a-for-loop-into-a-while-loop/)

Iterate over a String.

For loop version:

|  |  |
| --- | --- |
| 1  2  3 | str = 'Python'  for letter in str:      print letter |

While loop version:

|  |  |
| --- | --- |
| 1  2  3  4  5 | str = 'Python'  i = 0  while i < len(str):      print str[i]      i = i+1 |

In the for-loop we don’t need the index variable to iterate over the string, while in the while loop we use the variable i as the index for the string. We first initialize the variable i and set it to 0. Then, every time when the loop is executed, the variable i is increased by 1 until it reaches the last index.

## [Example 11: Sum up the digits in an integer](http://www.pd4cs.org/example-11-sum-up-the-digits-in-an-integer/)

Given an integer, say 287591, we want to calculate the sum of all its digits. The result would be 32 = 2+8+7+5+9+1.  This problem was already considered in the [**while-loop section (example 6)**](http://www.pd4cs.org/?p=665). Now we are solving it with a for-loop.

How do we proceed with a for-loop?  We change the integer into a string! Python allows us to iterate over the characters in the string and while doing so we add up “the digits.” Note that in the for-loop we need to convert the character “5” into digit 5 before making the addition.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # 02Loops example\_11 SumDigits2.py  # add up the digits in an integer using a for loop    n = int(raw\_input("Please input an integer:"))    sum = 0  for i in str(n):       #Convert integer sum into a string      #Convert the string reresenting a single digit into an integer and add      sum = sum + int(i)    print(sum) |

## [Example 12: Find the greatest common divisor (GCD) of two integers.](http://www.pd4cs.org/example-12-find-the-greatest-common-divisor-gcd-of-two-integers/)

Given two integers, the GCD is the largest positive integer that divides the numbers without a remainder. For example, the GCD of 32 and 24 is 8. Our final loop example is probably not suited for all students. It is a great example for a math-oriented teacher with mathematically more mature students. Students should have seen Euclid’s algorithm for finding the greatest common divisor of two integers. Euclid’s algorithm is an efficient method to compute the greatest common divisor. More detail is in [http://en.wikipedia.org/wiki/Euclidean\_algorithm.](http://en.wikipedia.org/wiki/Euclidean_algorithm) Many other good on-line sources exist.

To compute the gcd of 32 and 24 using Euclid’s algorithm:

* Divide 32 by 24 to get a quotient of 1 and a remainder of 8.
* Then divide 24 by 8 to get a quotient of 3 and a remainder of 0.
* 8 is the gcd of 32 and 24.

A program implementing Euclid’s algorithm is given below. It uses a while loop as it is not known how many iterations are needed. The algorithm has found the gcd when b=0.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | # 02Loops example\_12 GCDWhileLoop.py  # Euclid's algorithm for computing the gcd    a = input('Please input the first integer:')  b = input('Please input the second integer:')  while b != 0:      gcd = b      b = a % b      a = gcd  print(gcd) |

The three statements in the while loop are repeated until b becomes 0. The variable a is the dividend and the variable b is the divisor. According to the Euclid’s algorithm, the divisor in the previous step will be the dividend of the next step and the remainder of the previous step will be the divisor in next step. We use the variable gcd to store the divisor. When the variable b becomes 0, the value stored in gcd is the greatest common divisor of the two integers.

A conceptually simpler way for finding the greatest common divisor of two integers a and b is to try out all numbers from 1 to b:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | # 02Loops example\_12 GCDForLoop.py  # finding the gcd using a "brute force" iteration trying all values    a = input('Please input the first integer:')  b = input('Please input the second integer:')    for i in range(1,b+1):      if a % i == 0 and b % i == 0:          gcd = i  print(gcd) |

We divide both a and b by every integer from 1 to b. If both divide by the integer i, set gcd = i. When the loop ends, the value stored in gcd is the greatest common divisor of the two integers. Since 1 is always a common divisor, variable gcd will always have a value when the loop terminates.