wtech
Teknolojide Kadın Derneği

Introduction to Machine Learning with Python

`while` loops, miscellanous properties

Dr. Süha Tuna
İTÜ Informatics Institute

Introductory Examples

Example 1 Let's go back to the first program we wrote back in Section 1.3, the temperature converter. One annoying thing about it is that the user has to restart the program for every new temperature. A while loop will allow the user to repeatedly enter temperatures. A simple way for the user to indicate that they are done is to have them enter a nonsense temperature like −1000 (which is below absolute 0). This is done below:

```
temp = 0
while temp!=-1000:
    temp = eval(input('Enter a temperature (-1000 to quit): '))
    print('In Fahrenheit that is', 9/5*temp+32)
```

Introductory Examples

Example 2 One problem with the previous program is that when the user enters in −1000 to quit, the program still converts the value −1000 and doesn't give any message to indicate that the program has ended. A nicer way to do the program is shown below.

```
temp = 0
while temp!=-1000:
    temp = eval(input('Enter a temperature (-1000 to quit): '))
    if temp!=-1000:
        print('In Fahrenheit that is', 9/5*temp+32)
    else:
        print('Bye!')
```

Example 3 When first met if statements in Section 4.1, we wrote a program that played a simple random number guessing game. The problem with that program is that the player only gets one guess. We can, in a sense, replace the if statement in that program with a while loop to create a program that allows the user to keep guessing until they get it right.

```
from random import randint
secret_num = randint(1,10)
guess = 0
while guess != secret_num:
    guess = eval(input('Guess the secret number: '))
print('You finally got it!')
```

Introductory Examples

Example 4 We can use a while loop to mimic a for loop, as shown below. Both loops have the exact same effect.

```
for i in range(10):
    print(i)
```

```
i=0
while i<10:
    print(i)
    i=i+1
```

Remember that the for loop starts with the loop variable `i` equal to 0 and ends with it equal to 9. To use a while loop to mimic the for loop, we have to manually create our own loop variable `i`. We start by setting it to 0. In the while loop we have the same `print` statement as in the for loop, but we have another statement, `i=i+1`, to manually increase the loop variable, something that the for loop does automatically.

Example 5 Below is our old friend that converts from Fahrenheit to Celsius.

```
temp = eval(input('Enter a temperature in Celsius: '))
print('In Fahrenheit, that is', 9/5*temp+32)
```

A program that gets input from a user may want to check to see that the user has entered valid data. The smallest possible temperature is absolute zero, -273.15 °C. The program below takes absolute zero into account:

```
temp = eval(input('Enter a temperature in Celsius: '))
if temp<-273.15:
    print('That temperature is not possible.'))
else:
    print('In Fahrenheit, that is', 9/5*temp+32)
```

Infinite loops

When working with while loops, sooner or later you will accidentally send Python into a never-ending loop. Here is an example:

```
i=0
while i<10:
    print(i)
```

In this program, the value of `i` never changes and so the condition `i<10` is always true. Python will continuously print zeroes. To stop a program caught in a never-ending loop, use `Restart Shell` under the `Shell` menu. You can use this to stop a Python program before it is finished executing.

Sometimes a never-ending loop is what you want. A simple way to create one is shown below:

```
while True:
    # statements to be repeated go here
```

The break statement

Example 1 Here is a program that allows the user to enter up to 10 numbers. The user can stop early by entering a negative number.

```
for i in range(10):
    num = eval(input('Enter number: '))
    if num<0:
        break
```

This could also be accomplished with a while loop.

```
i=0
num=1
while i<10 and num>0:
    num = eval(input('Enter a number: '))
```

Either method is ok. In many cases the `break` statement can help make your code easier to understand and less clumsy.

Example 2 Earlier in the chapter, we used a while loop to allow the user to repeatedly enter temperatures to be converted. Here is, more or less, the original version on the left compared with a different approach using the `break` statement.

```
temp = 0
while temp!=-1000:
    temp = eval(input(': '))
    if temp!=-1000:
        print(9/5*temp+32)
    else:
        print('Bye!')
```

```
while True:
    temp = eval(input(': '))
    if temp==-1000:
        print('Bye')
        break
    print(9/5*temp+32)
```

The `else` statement

There is an optional `else` that you can use with `break` statements. The code indented under the `else` gets executed only if the loop completes without a `break` happening.

Example 1 This is a simple example based off of Example 1 of the previous section.

```
for i in range(10):
    num = eval(input('Enter number: '))
    if num<0:
        print('Stopped early')
        break
else:
    print('User entered all ten values')
```

Example 2 Here are two ways to check if an integer `num` is prime. A prime number is a number whose only divisors are 1 and itself. The approach on the left uses a while loop, while the approach on the right uses a for/break loop:

```
i=2
while i<num and num%i!=0:
    i=i+1
if i==num:
    print('Prime')
else:
    print('Not prime')
```

```
for i in range(2, num):
    if num%i==0:
        print('Not prime')
        break
else:
    print('Prime')
```

Exercises

1. The code below prints the numbers from 1 to 50. Rewrite the code using a while loop to accomplish the same thing.

```
for i in range(1,51):
    print(i)
```

2. (a) Write a program that uses a while loop (not a for loop) to read through a string and print the characters of the string one-by-one on separate lines.

   (b) Modify the program above to print out every second character of the string.

3. A good program will make sure that the data its users enter is valid. Write a program that asks the user for a weight and converts it from kilograms to pounds. Whenever the user enters a weight below 0, the program should tell them that their entry is invalid and then ask them again to enter a weight. [Hint: Use a while loop, not an if statement].

4. Write a program that asks the user to enter a password. If the user enters the right password, the program should tell them they are logged in to the system. Otherwise, the program should ask them to reenter the password. The user should only get five tries to enter the password, after which point the program should tell them that they are kicked off of the system.

5. Write a program that allows the user to enter any number of test scores. The user indicates they are done by entering in a negative number. Print how many of the scores are A's (90 or above). Also print out the average.

Exercises

6. Modify the higher/lower program so that when there is only one guess left, it says `1 guess`, not `1 guesses`.

7. Recall that, given a string `s`, `s.index('x')` returns the index of the first x in `s` and an error if there is no x.

   (a) Write a program that asks the user for a string and a letter. Using a while loop, the program should print the index of the first occurrence of that letter and a message if the string does not contain the letter.

   (b) Write the above program using a for/break loop instead of a while loop.

8. The GCD (greatest common divisor) of two numbers is the largest number that both are divisible by. For instance, gcd(18, 42) is 6 because the largest number that both 18 and 42 are divisible by is 6. Write a program that asks the user for two numbers and computes their gcd. Shown below is a way to compute the GCD, called Euclid's Algorithm.

   - First compute the remainder of dividing the larger number by the smaller number
   - Next, replace the larger number with the smaller number and the smaller number with the remainder.
   - Repeat this process until the smaller number is 0. The GCD is the last value of the larger number.

str, int, float, and list

The `str`, `int`, `float`, and `list` functions are used to convert one data type into another.

`str` Quite often we will want to convert a number to a string to take advantage of string methods to break the number apart. The built-in function `str` is used to convert things into strings. Here are some examples:

Statement	Result
`str(37)`	`'37'`
`str(3.14)`	`'3.14'`
`str([1,2,3])`	`'[1,2,3]'`

`int` and `float` The `int` function converts something into an integer. The `float` function converts something into a floating point number. Here are some examples.

Statement	Result
`int('37')`	37
`float('3.14')`	3.14
`int(3.14)`	3

To convert a float to an integer, the `int` function drops everything after the decimal point.

`list` The `list` function takes something that can be converted into a list and makes into a list. Here are two uses of it.

```
list(range(5))   [0,1,2,3,4]
list('abc')      ['a','b','c']
```

Examples

Example 1 Here is an example that finds all the palindromic numbers between 1 and 10000. A palindromic number is one that is the same backwards as forwards, like 1221 or 64546.

```
for i in range(1,10001):
    s = str(i)
    if s==s[::-1]:
        print(s)
```

We use the `str` function here to turn the integer `i` into a string so we can use slices to reverse it.

Example 2 Here is an example that tells a person born on January 1, 1991 how old they are in 2010.

```
birthday = 'January 1, 1991'
year = int(birthday[-4:])
print('You are', 2010-year, 'years old.')
```

The year is in the last four characters of `birthday`. We use `int` to convert those characters into an integer so we can do math with the year.

Example 3 Write a program that takes a number `num` and adds its digits. For instance, given the number 47, the program should return 11 (which is 4 + 7). Let us start with a 2-digit example.

```
digit = str(num)
answer = int(digit[0]) + int(digit[1])
```

The idea here is that we convert `num` to a string so that we can use indexing to get the two digits separately. We then convert each back to an integer using the `int` function. Here is a version that handles numbers with arbitrarily many digits:

```
digit = str(num)
answer = 0
for i in range(len(digit)):
    answer = answer + int(digit[i])
```

Examples

We can do the above program in a single line using a list comprehension.

```
answer = sum([int(c) for c in str(num)])
```

Example 4 To break a decimal number, `num`, up into its integer and fractional parts, we can do the following:

```
ipart = int(num)
dpart = num - int(num)
```

For example, if `num` is 12.345, then `ipart` is 12 and `dpart` is $12.345 - 12 = .345$.

Example 5 If we want to check to see if a number is prime, we can do so by checking to see if it has any divisors other than itself and 1. In Section 9.4 we saw code for this, and we had the following for loop:

```
for i in range(2,num):
```

This checks for divisibility by the integers 2, 3, ..., `num`-1. However, it turns out that you really only have to check the integers from 2 to the square root of the number. For instance, to check if 111 is prime, you only need to check if it is divisible by the integers 2 through 10, as $\sqrt{111} \approx 10.5$. We could then try the following for loop:

```
for i in range(2,num**.5):
```

However, this gives an error, because `num**.5` might not be an integer, and the `range` function needs integers. We can use `int` to correct this:

```
for i in range(2,int(num**.5)+1):
```

The `+1` at the end is needed due to the `range` function not including the last value.

Booleans

Boolean variables in Python are variables that can take on two values, `True` and `False`. Here are two examples of setting Boolean variables:

```
game_over = True
highlight_text = False
```

Booleans can help make your programs more readable. They are often used as flag variables or to indicate options. Booleans are often used as conditions in if statements and while loops:

```
if game_over:
    print('Bye!')
```

Note the following equivalences:

```
if game_over:          ⇔  if game_over==True:
while not game_over:   ⇔  while game_over==False:
```

note Conditional expressions evaluate to booleans and you can even assign them to variables. For instance, the following assigns `True` to `x` because `6==6` evaluates to `True`.

```
x = (6==6)
```

Shortcuts

- **Shortcut operators** Operations like `count=count+1` occur so often that there is a shorthand for them. Here are a couple of examples:

Statement	Shorthand
`count=count+1`	`count+=1`
`total=total-5`	`total-=5`
`prod=prod*2`	`prod*=2`

  There are also shortcut operators `/=`, `%=`, `//=`, and `**=`.

- **An assignment shortcut**

  Look at the code below.

```
a = 0
b = 0
c = 0
```

  A nice shortcut is:

```
a = b = c = 0
```

Shortcuts

- **Another assignment shortcut**

  Say we have a list `L` with three elements in it, and we want to assign those elements to variable names. We could do the following:

```
x = L[0]
y = L[1]
z = L[2]
```

  Instead, we can do this:

```
x,y,z = L
```

  Similarly, we can assign three variables at a time like below:

```
x,y,z = 1,2,3
```

  And, as we have seen once before, we can swap variables using this kind of assignment.

```
x,y,z = y,z,x
```

Here are some handy shortcuts:

Statement	Shortcut
`if a==0 and b==0 and c==0:`	`if a==b==c==0:`
`if 1<a and a<b and b<5:`	`if 1<a<b<5:`

Short-circuiting

Say we are writing a program that searches a list of words for those whose fifth character is `'z'`. We might try the following:

```
for w in words:
    if w[4]=='z':
        print(w)
```

But with this, we will occasionally get a `string index out of range` error. The problem is that some words in the list might be less than five characters long. The following if statement, however, will work:

```
if len(w)>=5 and w[4]=='z':
```

Continuation

Sometimes you'll write a long line of code that would be more readable if it were split across two lines. To do this, use a backslash \ character at the end of the line to indicate that the statement continues onto the next line. Here is an example:

```
if 'a' in string or 'b' in string or 'c' in string \
   or 'd' in string or 'e' in string:
```

Make sure there are no extra spaces after the backslash or you will get an error message.

If you are entering a list, dictionary, or the arguments of a function, the backslash can be left out:

```
L = ['Joe', 'Bob', 'Sue', 'Jimmy', 'Todd', 'Frank',
     'Mike', 'John', 'Amy', 'Edgar', 'Sam']
```

String formatting

Suppose we are writing a program that calculates a 25% tip on a bill of $23.60. When we multiply, we get 5.9, but we would like to display the result as `$5.90`, not `$5.9`. Here is how to do it:

```
a = 23.60 * .25
print('The tip is {:.2f}'.format(a))
```

This uses the `format` method of strings. Here is another example:

```
bill = 23.60
tip = 23.60*.25
print('Tip: ${:.2f}, Total: ${:.2f}'.format(tip, bill+tip))
```

The way the `format` method works is we put a pair of curly braces `{}` anywhere that we want a formatted value. The arguments to the format function are the values we want formatted, with the first argument matching up with the first set of braces, the second argument with the second set of braces, etc. Inside each set of curly braces you can specify a formatting code to determine how the corresponding argument will be formatted.

Formatting floats

Formatting integers To format integers, the formatting code is `{:d}`. Putting a number in front of the `d` allows us to right-justify integers. Here is an example:

```
print('{:3d}'.format(2))
print('{:3d}'.format(25))
print('{:3d}'.format(138))
```

```
  2
 25
138
```

The number 3 in these examples says that the value is allotted three spots. The value is placed as far right in those three spots as possible and the rest of the slots will be filled by spaces. This sort of thing is useful for nicely formatting tables.

To center integers instead of right-justifying, use the `^` character, and to left-justify, use the `<` character.

```
print('{:^5d}'.format(2))
print('{:^5d}'.format(222))
print('{:^5d}'.format(13834))
```

```
  2
 122
13834
```

Each of these allots five spaces for the integer and centers it within those five spaces.

Putting a comma into the formatting code will format the integer with commas. The example below prints `1,000,000`:

```
print('{:,d}'.format(1000000))
```

Formatting integers

Formatting floats To format a floating point number, the formatting code is `{:f}`. To only display the number to two decimal places, use `{:.2f}`. The `2` can be changed to change the number of decimal places.

You can right-justify floats. For example, `{:8.2f}` will allot eight spots for its value—one of those is for the decimal point and two are for the part of the value after the decimal point. If the value is `6.42`, then only four spots are needed and the remaining spots are filled by spaces, causing the value to be right-justified.

The `^` and `<` characters center and left-justify floats.

Formatting strings To format strings, the formatting code is `{:s}`. Here is an example that centers some text:

```
print('{:^10s}'.format('Hi'))
print('{:^10s}'.format('there!'))
```

```
   Hi
  there!
```

To right-justify a string, use the `>` character:

```
print('{:>6s}'.format('Hi'))
print('{:>6s}'.format('There'))
```

```
    Hi
there!
```

Nested loops

Example 1 Print a 10×10 multiplication table.

```
for i in range(1,11):
    for j in range(1,11):
        print('{:3d}'.format(i*j), end=' ')
    print()
```

A multiplication table is a two-dimensional object. To work with it, we use two for loops, one for the horizontal direction and one for the vertical direction. The print statement right justifies the products to make them look nice. The `end=''` allows us to print several things on each row. When we are done printing a row, we use `print()` to advance things to the next line.

Example 2 A common math problem is to find the solutions to a system of equations. Sometimes you want to find only the integer solutions, and this can be a little tricky mathematically. However, we can write a program that does a brute force search for solutions. Here we find all the integer solutions (x, y) to the system $2x + 3y = 4$, $x - y = 7$, where x and y are both between -50 and 50.

```
for x in range(-50,51):
    for y in range(-50,51):
        if 2*x+3*y==4 and x-y==7:
            print(x,y)
```

Nested loops

Example 3 A Pythagorean triple is a triple of numbers (x, y, z) such that $x^2 + y^2 = z^2$. For instance $(3, 4, 5)$ is a Pythagorean triple because $3^2 + 4^2 = 5^2$. Pythagorean triples correspond to triangles whose sides are all whole numbers (like a 3-4-5-triangle). Here is a program that finds all the Pythagorean triples (x, y, z) where x, y, and z are positive and less than 100.

```
for x in range(1,100):
    for y in range(1,100):
        for z in range(1,100):
            if x**2+y**2==z**2:
                print(x,y,z)
```

Example 5 Your computer screen is grid of pixels. To draw images to the screen, we often use nested for loops—one loop for the horizontal direction, and one for the vertical direction. See Sections 18.2 and 22.6 for examples.

Example 6 List comprehensions can contain nested for loops. The example below returns a list of all the vowels in a list of words.

```
[char for item in L for char in item if char in 'aeiou']
```

Exercises

1. Write a program that uses `list` and `range` to create the list `[3, 6, 9, ..., 99]`.

2. Write a program that asks the user for a weight in kilograms. The program should convert the weight to kilograms, formatting the result to one decimal place.

3. Write a program that asks the user to enter a word. Rearrange all the letters of the word in alphabetical order and print out the resulting word. For example, `abracadabra` should become `aaaaabbcdrr`.

4. Write a program that takes a list of ten prices and ten products, applies an 11% discount to each of the prices displays the output like below, right-justified and nicely formatted.

```
Apples     $  2.45
Oranges    $ 18.02
...
Pears      $120.03
```

5. Use the following two lists and the `format` method to create a list of card names in the format *card value* of *suit name* (for example, `'Two of Clubs'`).

```
suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']
values = ['One', 'Two', 'Three', 'Four', 'Five', 'Six', 'Seven',
          'Eight', 'Nine', 'Ten', 'Jack', 'Queen', 'King', 'Ace']
```

Exercises

6. Write a program that uses a boolean flag variable in determining whether two lists have any items in common.

7. Write a program that creates the list `[1,11,111,1111,...,111...1]`, where the entries have an ever increasing number of ones, with the last entry having 100 ones.

8. Write a program to find all numbers between 1 and 1000 that are divisible by 7 and end in a 6.

9. Write a program to determine how many of the numbers between 1 and 10000 contain the digit 3.

10. Adding certain numbers to their reversals sometimes produces a palindromic number. For instance, 241 + 142 = 383. Sometimes, we have to repeat the process. For instance, 84 + 48 = 132 and 132 + 231 = 363. Write a program that finds both two-digit numbers for which this process must be repeated more than 20 times to obtain a palindromic number.

11. Write a program that finds all pairs of six-digit palindromic numbers that are less than 20 apart. One such pair is 199991 and 200002.

12. The number 1961 reads the same upside-down as right-side up. Print out all the numbers between 1 and 100000 that read the same upside-down as right-side up.

13. The number 99 has the property that if we multiply its digits together and then add the sum of its digits to that, we get back to 99. That is, $(9 \times 9) + (9 + 9) = 99$. Write a program to find all of the numbers less than 10000 with this property. (There are only nine of them.)

14. Write a program to find the smallest positive integer that satisfies the following property: If you take the leftmost digit and move it all the way to the right, the number thus obtained is exactly 3.5 times larger than the original number. For instance, if we start with 2958 and move the 2 all the way to the right, we get 9582, which is roughly 3.2 times the original number.

15. Write a program to determine how many zeroes 1000! ends with.