Dealing with special characters in Python
Python gives special meaning to some characters in strings, most notably the backslash and tab characters and newline characters. We can print these characters using a backslash.

```
print ("A newline symbol \\n")
print ("A newline \n")
print ("A tab symbol \\t")
print ("A tab\t")
print ("A backlash symbol \\")
print ("A backslash \t")
```

Conditional statements

Conditional statements alter the flow of the program and act as decision points in the code. (The term "control flow" refers to the progression of logic/ tasks done and undone as the Python interpreter traverses the code). For example, one may only want to execute some commands in certain scenarios if a scenario is true.

A simple decision point is: **if** something, **do** something. The condition is some kind of test that must evaluate to either being true or false. For example, "if two nucleotides are within 100bp of each other, then analyze them together." "If the sequence is on chromosome 1, then print it."

The general structure is:

if condition**:**
  code to execute

#three things to note here are: the *if* statement, the *:*, and the *uniformly* indented code.

Conditions are evaluated as true or false. For example,

```
print (7==7)
print (7>5)
print (len("TAGC")>5)
print ("TAGCT".count("T")>=2)
print ("TAGCT".startswith("T"))
print ("TAGCT".endswith("A"))
print ("TAGCT".isupper())
print ("TAGCT".islower())
```

These statements are evaluated and assigned True or False. These are built-in values that represent true and false. For example, print (True) returns True; print (true) generates a NameError. "NameError: name 'true' is not defined"

For conditions, the basic building blocks are:

Equals (==)
Greater/Less than > and <
Greater/Less than and equal to >= and <=
Not equal !=

And others. As noted above, many methods return True or False values. We can evaluate if strings with the == and != comparitors.

Typical tasks are to do something if a condition is true; and do something if a condition is true and something different if a condition is not true.

if condition:
    do something
else:
    do something else

An example: Once again, we will obtain a function by importing it from a module with the import command. See https://docs.python.org/2/library/random.html

```
from random import randint #Here we are importing the
randint function to our current environment
a = randint (0, 100) #get a random number
print(a)
if (a<50):
    print (" is less than 50")
else:
    print (" is 50 or more")
```

The output of this code is a bit messy:

```
4
 is less than 50
```

The print command automatically adds a new line. We can stop this with

```
print(a, end ='')
```

Python may not interpret your code like you would expect because else is bound soley to the if statement in the same code block.

```
from random import randint #Here we are importing the
randint function to our current environment
a = randint (0, 100) #get a random number
print(a, end ='')
if (a<50):
    print (" is less than 50")
if (a>50):
    print (" is more than 50")
else:
```

```
    print (" is fifty exactly!")
```

An output of this is:
3 is less than 50
is fifty exactly!

We can fix this using `elif` which is else and if joined. The if/elif/else structure is useful when one has mutually exclusive options.

```
a = randint (0, 100) #get a random number
print(a, end ='')
if (a<50):
    print (" is less than 50")
elif (a>50):
    print (" is more than 50")
else:
    print (" is fifty exactly!")
```

If a scenario is not mutually exclusive- for example, you would want to keep track of numbers that are both greater than 50 and 75, one can use multiple if commands.

```
if (a>50):
    print (" It is more than 50")
if (a>75):
    print (" It is also more than 75")
```

Nested conditionals
Conditional statements can contain other conditional statements. In Python, the blocks of code inside the conditionals must be aligned.

```
#conditional.py
from random import randint
x= randint(1,7)
y= randint(1,7)

print ("x equals " + str(x) + " and y equals " + str(y))

if (x==y and x!=7):  # start of block of code.
     print ("MATCH x is equal to y!")
elif (x==7 and y==7):
     print ("My lucky number was matched!")

     if (x==y): #this is a nested conditonal
          print ("I am very lucky. Both x and y are 7s!")

else:
     print ("No match- Sorry!")
```

Note the use of “and” above and “or” also works.
Here is another example.

```
from random import randint
x=randint(1,20)
print ("x is " + str(x))

if x<10:
     print ("x has a single digit")
     if (x%2 ==0):
          print ("x is an even number")
```

#can you think of ways to modifying this program based on other conditions?

We can make conditions quite complex using and and or. We can also negate conditions with not.

```
from random import randint
x= randint(1,7)
y= randint(1,7)

print ("x equals " + str(x) + " and y equals " + str(y))

if ((x==y and x==7) or (x>5 or not y<6)):
     print ("MATCH x and y are seven, or x or y is greater
than five")
else:
     print ("No match- Sorry!")
```

Numeric precision and conditionals

Numbers are stored as either integers or floating point (decimal) numbers.
Float numbers do not have the exact number you expect. 0.1 is not exactly 1/10th.

```
y=0.1+0.1+0.1
x=0.3
print (x-y)
```

Never ask if two floats ==. Use abs() to determine if absolute difference is smaller than a threshhold.

```
y=0.1+0.1+0.1
x=0.3
threshhold=0.0001;
if ((abs(x-y)) < threshhold):
     print ("close enough!")
```

Stopping programs

You may want to stop a program because something went wrong. For example, a user inputted the wrong type of data, or a variable's value does not look ok.

```
from sys import exit #we will use the exit function
from random import randint
price = randint(1,100)
print ("The price is " + str(price))
if (price >50):
     exit ("Too much- No way!")
#we are not even considering the rest of the code.

print ("I had a 100 dollars, after I buy that I will have "
+ str(100-price) +" left")
```

Or, here is another example

```
from sys import exit
dna="ATGCTAGC"
length=len(dna)
if (length%3 !=0):
     exit ("Bad length")
print ("Good sequence")
```