Introduction to Programming Using Python

Microsoft Python Certification | **Exam 98-381**

Lecture 4:

Conditionals and Loops

Lecturer: **James W. Jiang**, Ph.D. | Summer, 2018

Microsoft™

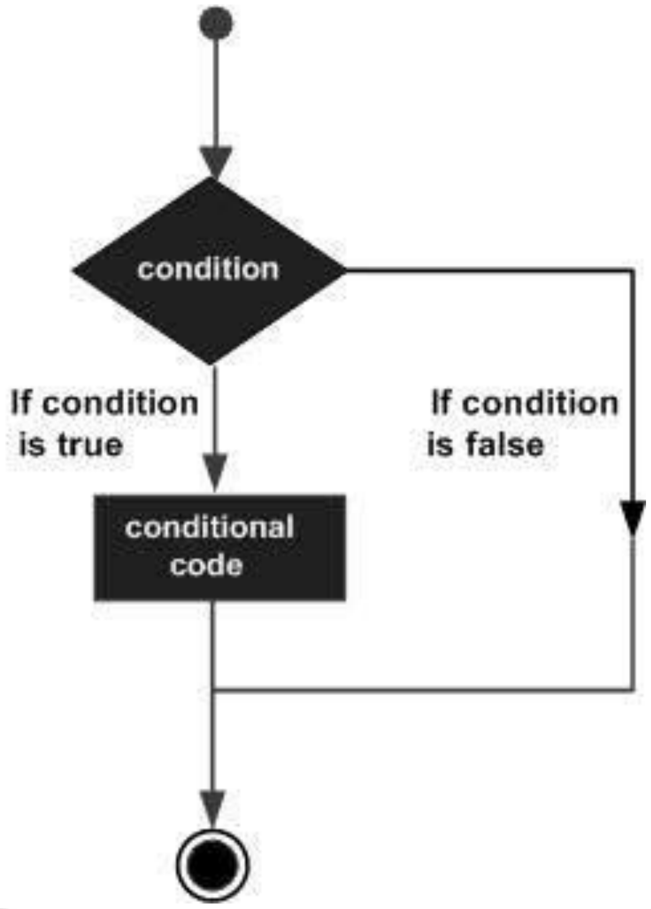
SAVVY PRO
PROFESSIONAL EDUCATION TRAINING

Conditionals:
One Conditional

Conditional Execution

The Boolean expression after `if` is called the condition. If it is true, the indented statement runs. If not, nothing happens.

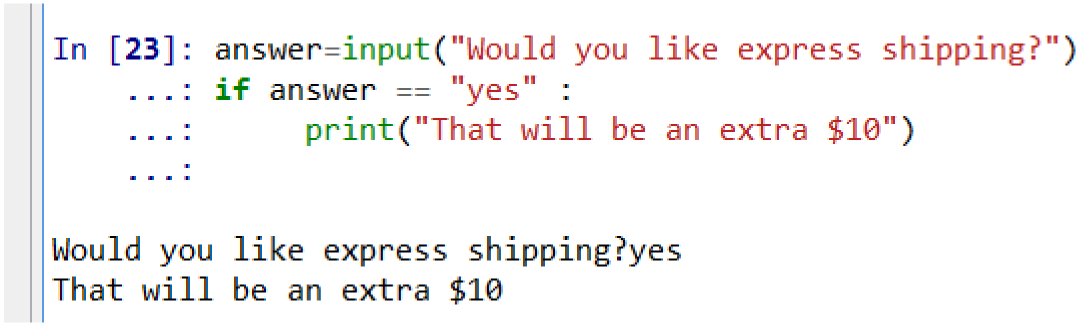
```
if x > 0:
    print('x is positive')
```



Conditional Execution

If statements allow you to specify code that only executes if a specific condition is true

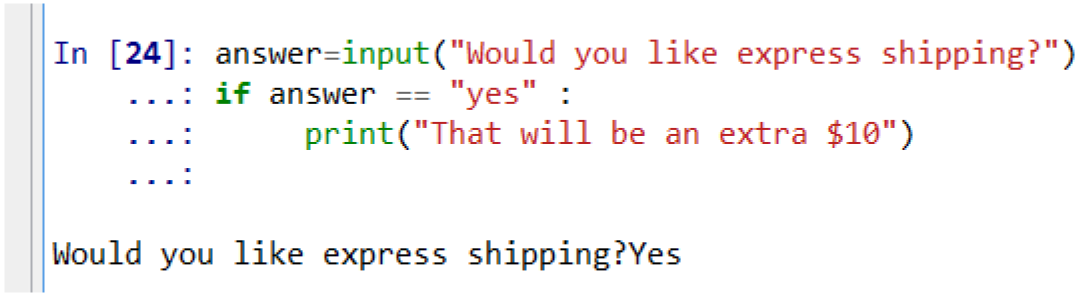
```
answer=input("Would you like express shipping?")
if answer == "yes" :
     print("That will be an extra $10")
```



```
In [23]: answer=input("Would you like express shipping?")
    ...: if answer == "yes" :
    ...:     print("That will be an extra $10")
    ...:

Would you like express shipping?yes
That will be an extra $10
```

Conditional Execution

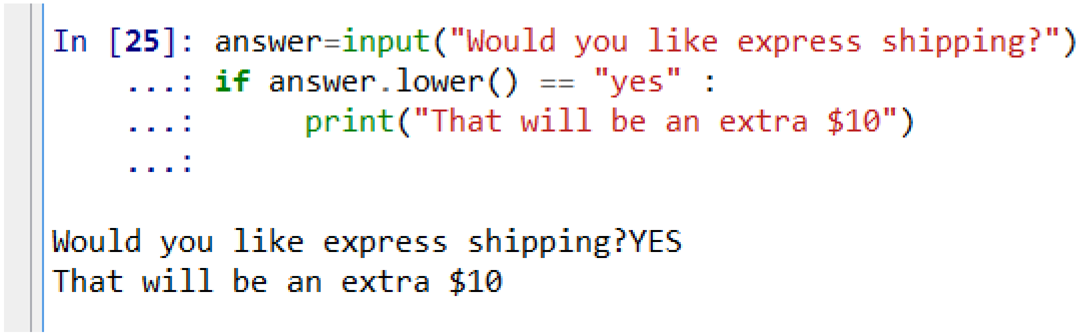
```
answer=input("Would you like express shipping?")
if answer == "yes" :
     print("That will be an extra $10")
```



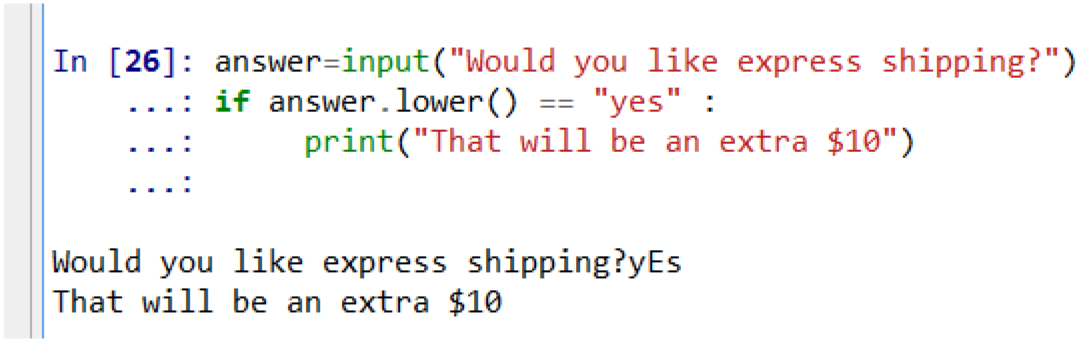
```
In [24]: answer=input("Would you like express shipping?")
    ...: if answer == "yes" :
    ...:      print("That will be an extra $10")
    ...:

Would you like express shipping?Yes
```

Conditional Execution

```
answer=input("Would you like express shipping?")
if answer.lower() == "yes" :
    print("That will be an extra $10")
```



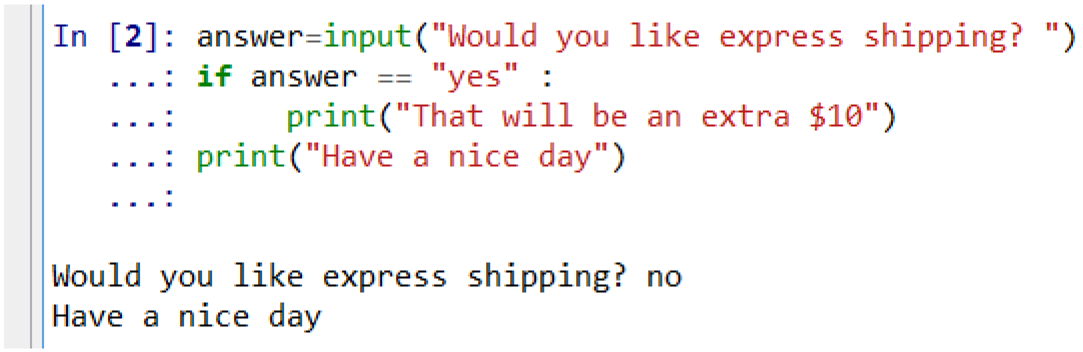
```
In [25]: answer=input("Would you like express shipping?")
    ...: if answer.lower() == "yes" :
    ...:     print("That will be an extra $10")
    ...:

Would you like express shipping?YES
That will be an extra $10
```



```
In [26]: answer=input("Would you like express shipping?")
    ...: if answer.lower() == "yes" :
    ...:     print("That will be an extra $10")
    ...:

Would you like express shipping?yEs
That will be an extra $10
```

Conditional Execution

```
answer=input("Would you like express shipping? ")
if answer == "yes" :
     print("That will be an extra $10")
print("Have a nice day")
```



```
In [2]: answer=input("Would you like express shipping? ")
   ...: if answer == "yes" :
   ...:     print("That will be an extra $10")
   ...: print("Have a nice day")
   ...:

Would you like express shipping? no
Have a nice day
```

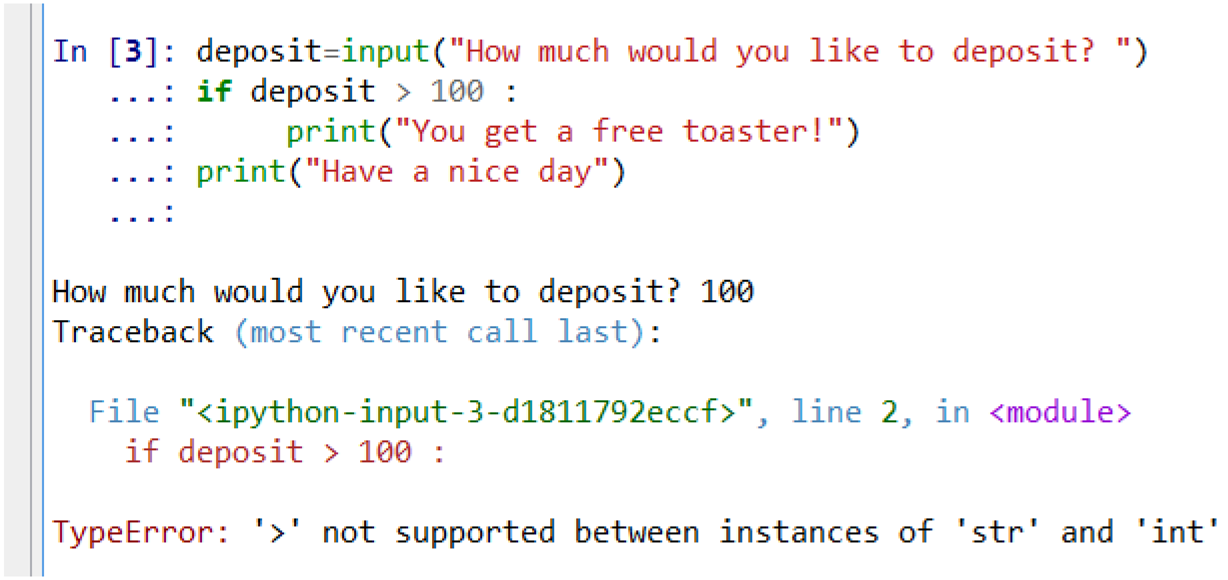
PROFESSIONAL EDUCATION TRAINING

Conditionals:

Numerical Input

Handling Numerical Inputs

```
deposit=input("How much would you like to deposit? ")
if deposit > 100 :
     print("You get a free toaster!")
print("Have a nice day")
```

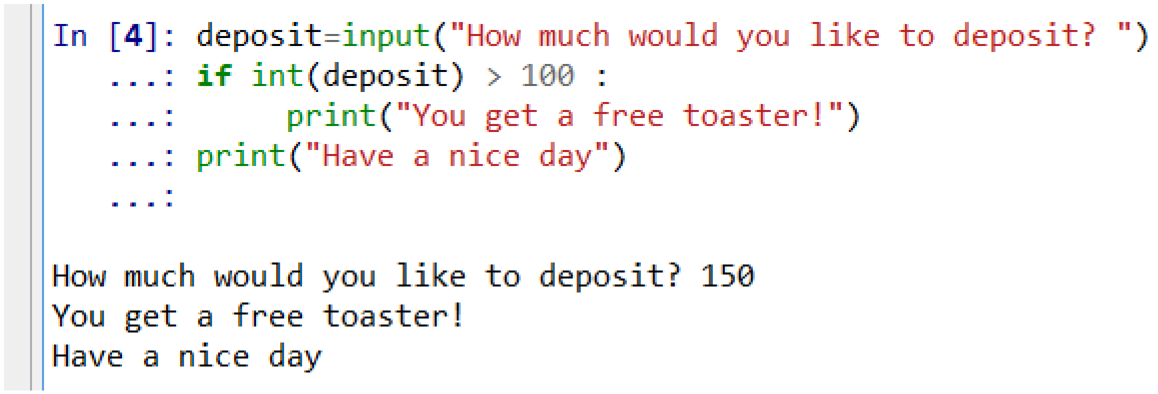


```
In [3]: deposit=input("How much would you like to deposit? ")
   ...: if deposit > 100 :
   ...:      print("You get a free toaster!")
   ...: print("Have a nice day")
   ...:

How much would you like to deposit? 100
Traceback (most recent call last):

  File "<ipython-input-3-d1811792eccf>", line 2, in <module>
    if deposit > 100 :

TypeError: '>' not supported between instances of 'str' and 'int'
```

Handling Numerical Inputs

We have to convert the string value returned by the input function to a number

```
deposit=input("How much would you like to deposit? ")
if int(deposit) > 100 :
     print("You get a free toaster!")
print("Have a nice day")
```

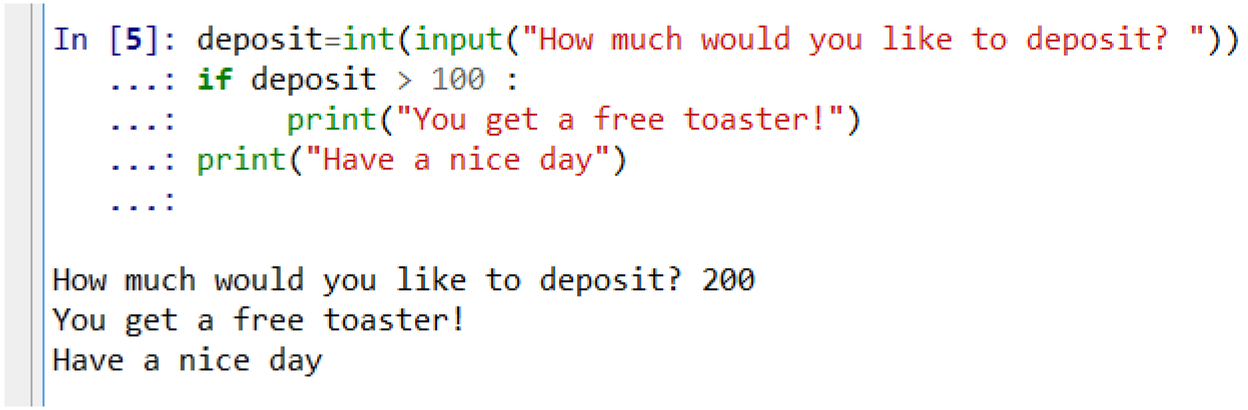


```
In [4]: deposit=input("How much would you like to deposit? ")
   ...: if int(deposit) > 100 :
   ...:      print("You get a free toaster!")
   ...: print("Have a nice day")
   ...:

How much would you like to deposit? 150
You get a free toaster!
Have a nice day
```

Handling Numerical Inputs

We have to convert the string value returned by the input function to a number

```
deposit=int(input("How much would you like to deposit?
"))
if deposit > 100 :
     print("You get a free toaster!")
print("Have a nice day")
```



```
In [5]: deposit=int(input("How much would you like to deposit? "))
   ...: if deposit > 100 :
   ...:     print("You get a free toaster!")
   ...: print("Have a nice day")
   ...:

How much would you like to deposit? 200
You get a free toaster!
Have a nice day
```

Almost every if statement can be written two ways

```
if answer == "yes" :
if not answer == "no" :

if total < 100 :
if not total >= 100 :
```

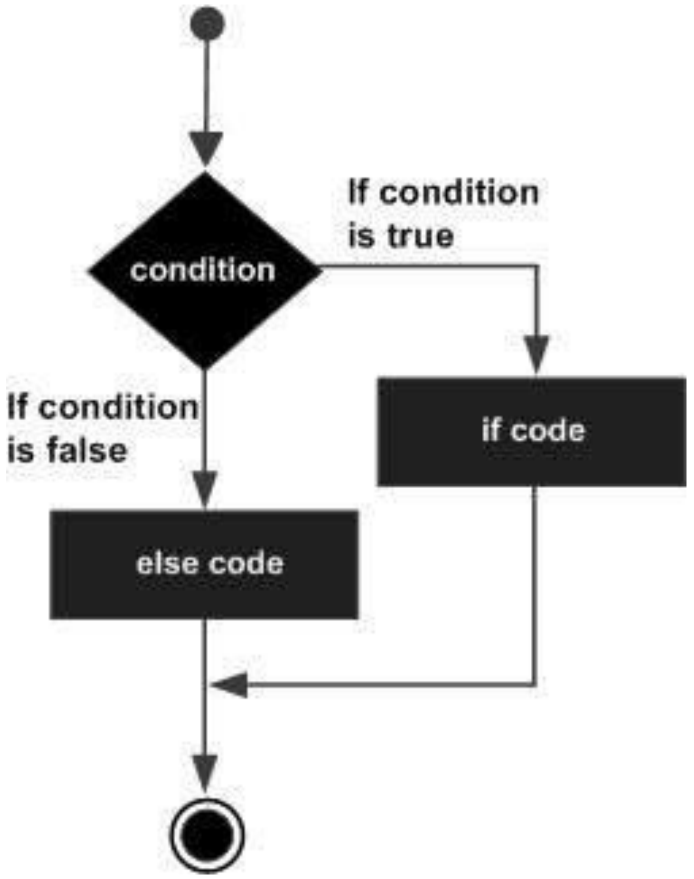
Which do you prefer?

Conditionals:
Multiple Conditionals

Alternative Execution

A second form of the if statement is “alternative execution”, in which there are two possibilities and the condition determines which one runs.

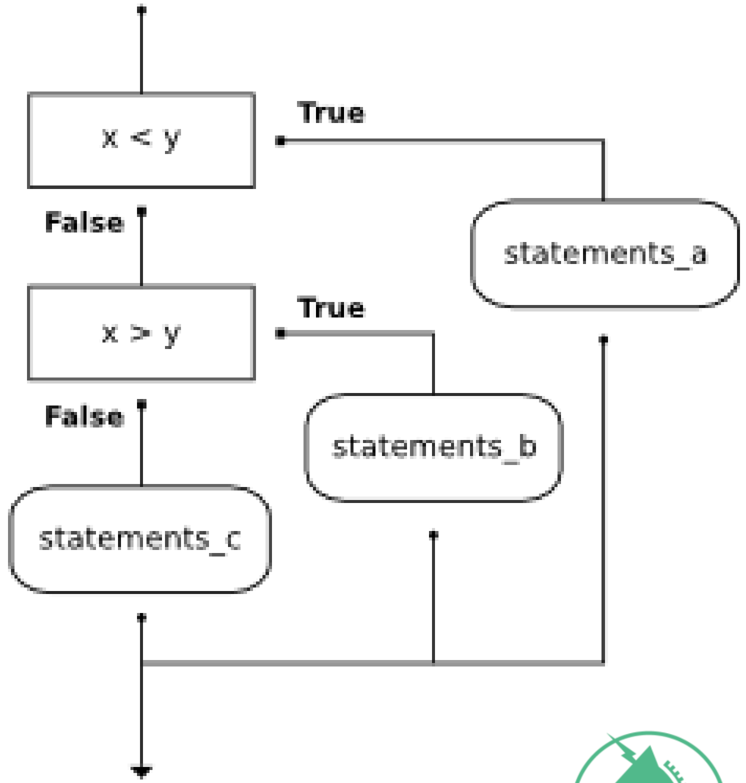
```
if x % 2 == 0:
    print('x is even')
else:
    print('x is odd')
```



Chained Conditionals

Sometimes there are more than two possibilities and we need more than two branches. One way to express a computation like that is a chained conditional

```
if x < y :
    print('x is less than y')
elif x > y :
    print('x is greater than y')
else:
    print('x and y are equal')
```

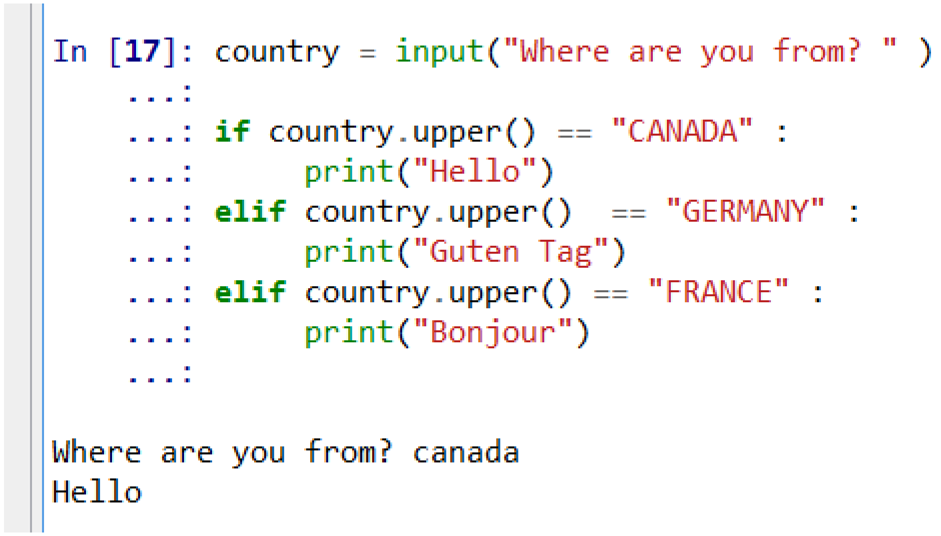


if, elif

```
country = input("Where are you from? " )

if country == "CANADA" :
     print("Hello")
elif country == "GERMANY" :
     print("Guten Tag")
elif country == "FRANCE" :
     print("Bonjour")
```

if, elif, elif

```
country = input("Where are you from? " )

if country.upper() == "CANADA" :
     print("Hello")
elif country.upper()  == "GERMANY" :
     print("Guten Tag")
elif country.upper() == "FRANCE" :
     print("Bonjour")
```



```
In [17]: country = input("Where are you from? " )
    ...:
    ...: if country.upper() == "CANADA" :
    ...:      print("Hello")
    ...: elif country.upper()  == "GERMANY" :
    ...:      print("Guten Tag")
    ...: elif country.upper() == "FRANCE" :
    ...:      print("Bonjour")
    ...:

Where are you from? canada
Hello
```

if, elif, ..., else

We should add an “**else**” statement to catch any conditions we didn’t list

```
country = input("Where are you from? " )

if country.upper() == "CANADA" :
     print("Hello")
elif country.upper() == "GERMANY" :
     print("Guten Tag")
elif country.upper() == "FRANCE" :
     print("Bonjour")
else :
     print("Aloha/Ciao/G’Day/Ni Hao")
```

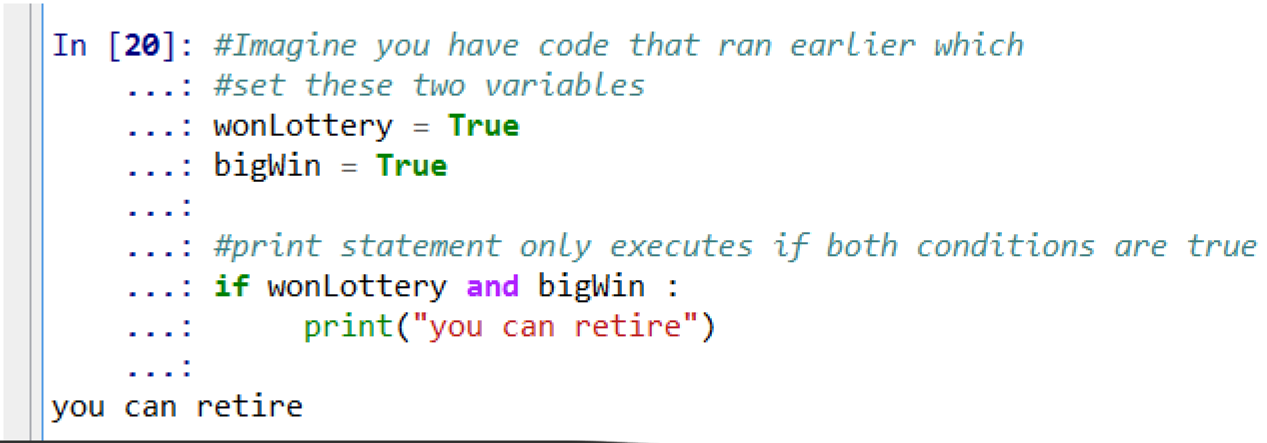
Conditionals:
Combining Conditions

Types of Conditions

```
if 0 < x:
    if x < 10:
    print('x is a positive single-digit number.’)

if 0 < x and x < 10:
    print('x is a positive single-digit number.’)

if 0 < x < 10:
    print('x is a positive single-digit number.')
```

And

```
#Imagine you have code that ran earlier which
#set these two variables
wonLottery = True
bigWin = True
#print statement only executes if both conditions are tr
ue
if wonLottery and bigWin :
     print("you can retire")
```



```
In [20]: #Imagine you have code that ran earlier which
    ...: #set these two variables
    ...: wonLottery = True
    ...: bigWin = True
    ...:
    ...: #print statement only executes if both conditions are true
    ...: if wonLottery and bigWin :
    ...:     print("you can retire")
    ...:
you can retire
```

And

```
if firstCondition and secondCondition :
```

First Condition is	Second Condition is	Statement is
`True`	`True`	`True`
`True`	`False`	`False`
`False`	`True`	`False`
`False`	`False`	`False`

or

```
#Imagine you have code that ran earlier which
#set these two variables
saturday = True
sunday = False

#print statement executes if either condition is true
if saturday or sunday :
     print("you can sleep in")
```

or

```
if firstCondition or secondCondition :
```

First Condition is	Second Condition is	Statement is
True	True	True
True	False	True
False	True	True
False	False	False

Conditionals:
Multiple And(s) and Or(s)

Multiple or(s)

```
if month == "Sep" or month =="Apr" \
      or month == "Jun" or month == "Nov" :
      print("There are 30 days in this month")
```

Multiple and(s)

```
if favMovie == "Star Wars" \
      and favBook == "Lord of the Rings" \
      and favEvent == "ComiCon" :
      print("You and I should hang out")
```

Multiple or(s)/and(s)

When in doubt, just add parentheses whenever you combine and/or in a single if statement.

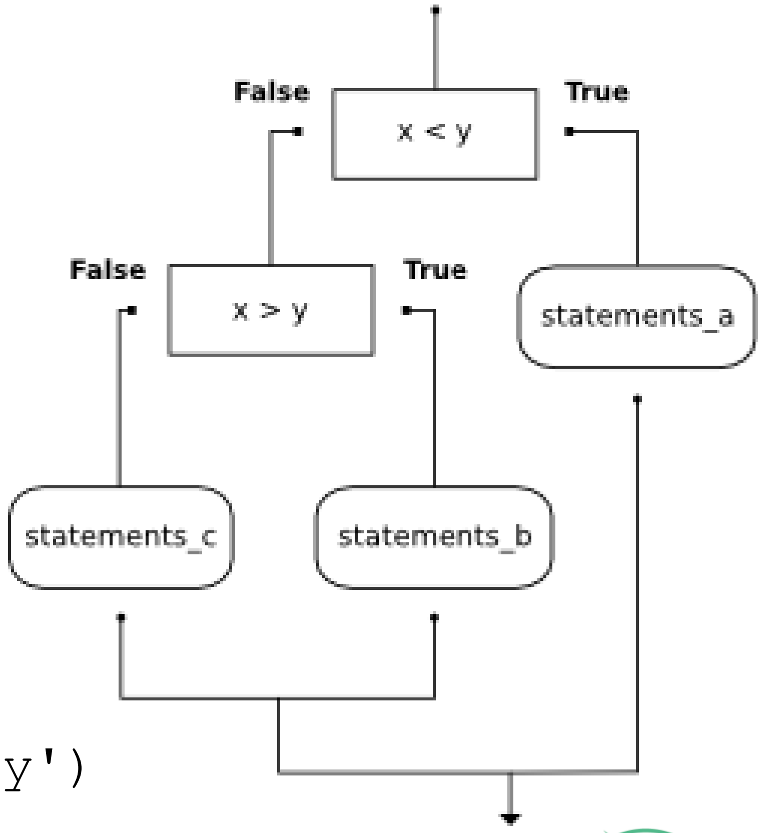
```
if country == "CANADA" and \
      (pet == "MOOSE" or  pet == "BEAVER") :
      print("Do you play hockey too")
```

Conditionals:
Nested Conditionals

Nested Conditionals

One conditional can also be nested within another. We could have written the example in the previous section like this:

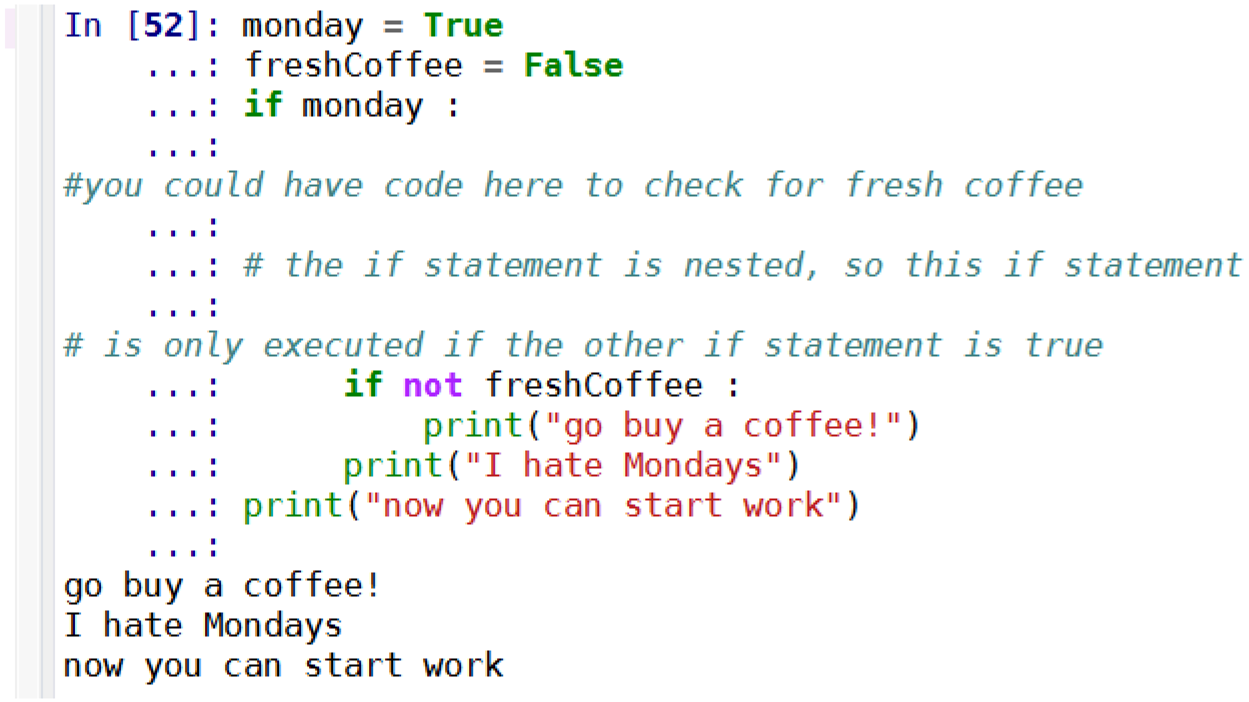
```
if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y’)
    else:
        print('x is greater than y')
```



Nested Conditionals

```
monday = True
freshCoffee = False
if monday :
#you could have code here to check for fresh coffee

# the if statement is nested, so this if statement
# is only executed if the other if statement is true
     if not freshCoffee :
         print("go buy a coffee!")
     print("I hate Mondays")
print("now you can start work")
```

Nested Conditionals



```
In [52]: monday = True
    ...: freshCoffee = False
    ...: if monday :
    ...:
#you could have code here to check for fresh coffee
    ...:
    ...: # the if statement is nested, so this if statement
    ...:
# is only executed if the other if statement is true
    ...:        if not freshCoffee :
    ...:            print("go buy a coffee!")
    ...:        print("I hate Mondays")
    ...: print("now you can start work")
    ...:
go buy a coffee!
I hate Mondays
now you can start work
```

PROFESSIONAL EDUCATION TRAINING

Conditionals:
Boolean Variables

Boolean Variables

You can use Boolean variables to remember if a condition is true or false

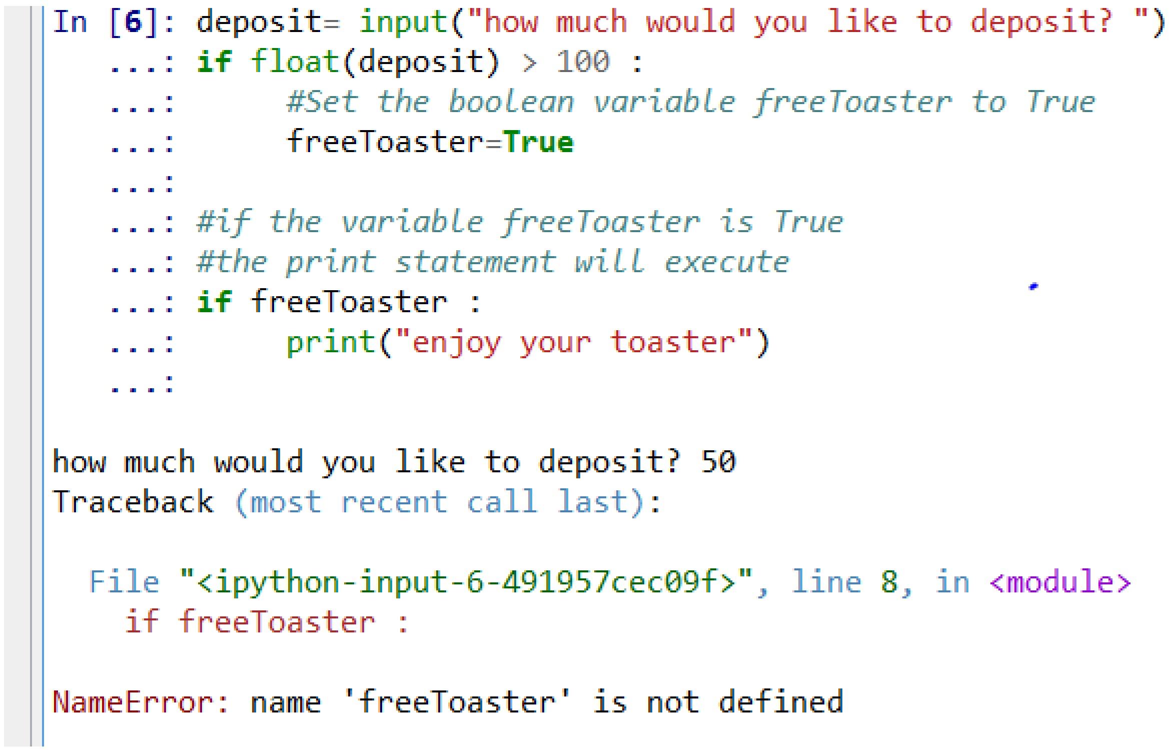
```
deposit= input("how much would you like to deposit? ")
if float(deposit) > 100 :
     #Set the boolean variable freeToaster to True
     freeToaster=True

#if the variable freeToaster is True
#the print statement will execute
if freeToaster :
     print("enjoy your toaster")
```

Boolean Variables

Why does our code crash when we enter a value of 50 for a deposit?

```
deposit= input("how much would you like to deposit? ")
if float(deposit) > 100 :
     #Set the boolean variable freeToaster to True
     freeToaster=True

#if the variable freeToaster is True
#the print statement will execute
if freeToaster :
     print("enjoy your toaster")
```

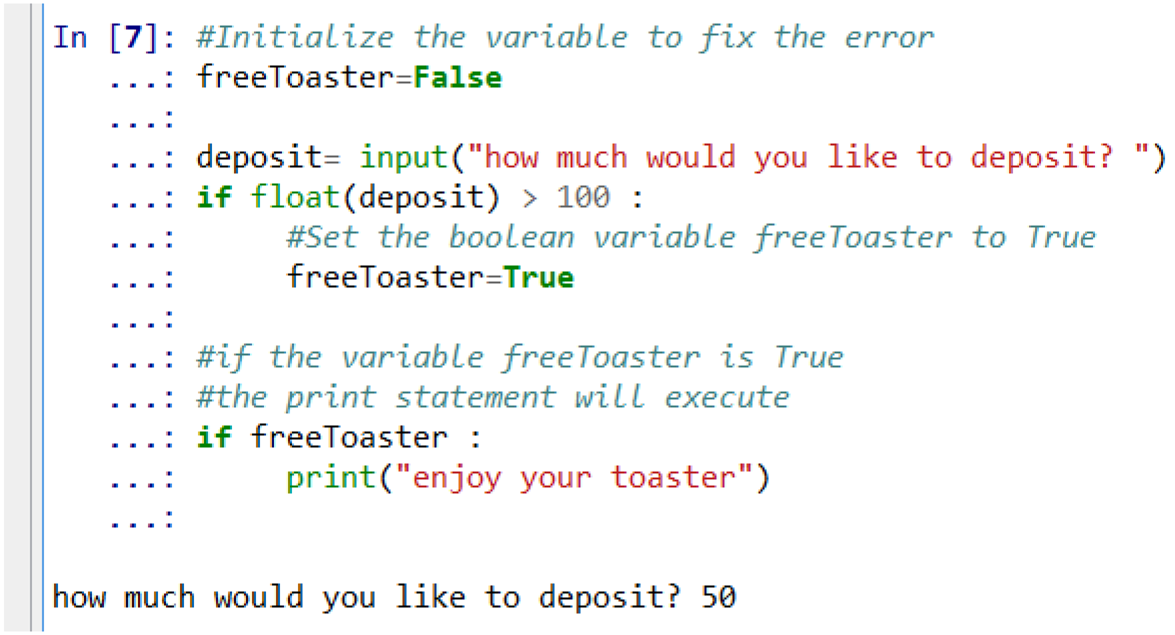
Boolean Variables


```
In [6]: deposit= input("how much would you like to deposit? ")
   ...: if float(deposit) > 100 :
   ...:     #Set the boolean variable freeToaster to True
   ...:     freeToaster=True
   ...:
   ...: #if the variable freeToaster is True
   ...: #the print statement will execute
   ...: if freeToaster :
   ...:     print("enjoy your toaster")
   ...:

how much would you like to deposit? 50
Traceback (most recent call last):

  File "<ipython-input-6-491957cec09f>", line 8, in <module>
    if freeToaster :

NameError: name 'freeToaster' is not defined
```

Initialize Variables

```
#Initialize the variable to fix the error
freeToaster=False

deposit= input("how much would you like to deposit? ")
if float(deposit) > 100 :
     #Set the boolean variable freeToaster to True
     freeToaster=True

#if the variable freeToaster is True
#the print statement will execute
if freeToaster :
     print("enjoy your toaster")
```

Initialize Variables



```
In [7]: #Initialize the variable to fix the error
   ...: freeToaster=False
   ...:
   ...: deposit= input("how much would you like to deposit? ")
   ...: if float(deposit) > 100 :
   ...:         #Set the boolean variable freeToaster to True
   ...:         freeToaster=True
   ...:
   ...: #if the variable freeToaster is True
   ...: #the print statement will execute
   ...: if freeToaster :
   ...:         print("enjoy your toaster")
   ...:

how much would you like to deposit? 50
```

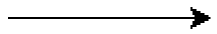
Loops:
Turtle

some of the turtle commands

Command	Action
`right(x)`	Rotate right x degrees
`left(x)`	Rotate left x degrees
`color('x')`	Change pen color to x
`forward(x)`	Move forward x
`backward(x)`	Move backward x

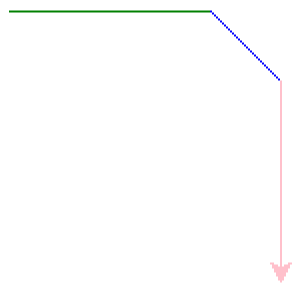
Did you know Python can draw?

```
import turtle
turtle.forward(100)

# close turtle window:
# turtle.Screen().bye()
```



.color(), .right(), .forward()

```
import turtle
turtle.color('green')
turtle.forward(100)
turtle.right(45)
turtle.color('blue')
turtle.forward(50)
turtle.right(45)
turtle.color('pink')
turtle.forward(100)
# turtle.Screen().bye()
```



Loops :
For Loops

Types of Loops

Loop	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another loop

Function: range

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers). We can also define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided.

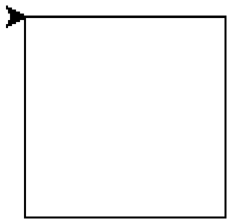
```
print(range(10)) # Output: range(0, 10)


print(list(range(10))) # Output: [0, 1, 2, 3, 4, 5, 6,
7, 8, 9]

print(list(range(2, 8))) # Output: [2, 3, 4, 5, 6, 7]


print(list(range(2, 20, 3))) # Output: [2, 5, 8, 11, 14,
17]
```

get turtle do draw a square

```
import turtle
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
```

Use for Loops

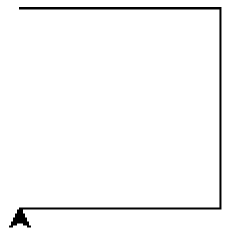
```
import turtle
for steps in range(4):
      turtle.forward(100)
      turtle.right(90)

# range(4): Number of times to execute the code in the
loop
```



Use for Loops

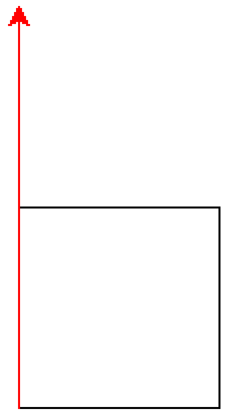
When you change the range, you change the number of times the code executes

```
import turtle
for steps in range(3):
      turtle.forward(100)
      turtle.right(90)
```



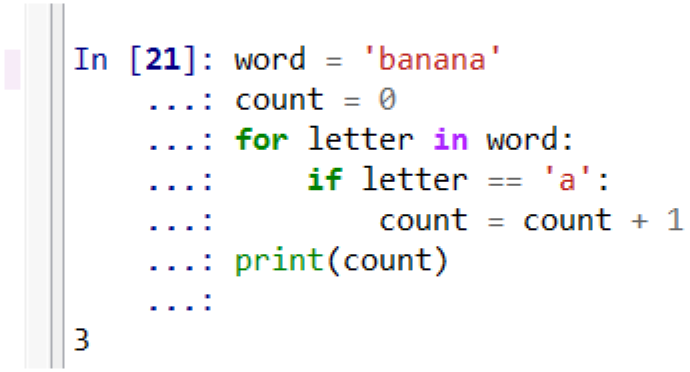
Only the indented code is repeated!

```
import turtle
for steps in range(3):
      turtle.forward(100)
      turtle.right(90)
turtle.color('red')
turtle.forward(200)
```



for and if

```
word = 'banana'
count = 0
for letter in word:
    if letter == 'a':
        count = count + 1
print(count)
```



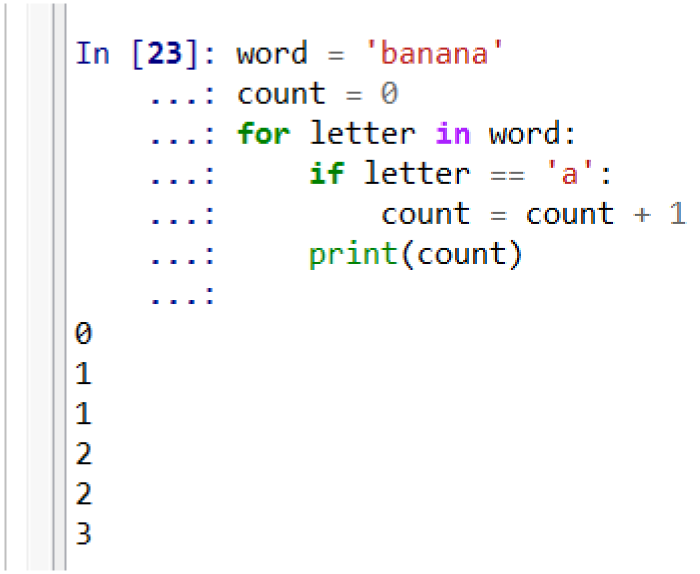
```
In [21]: word = 'banana'
    ...: count = 0
    ...: for letter in word:
    ...:     if letter == 'a':
    ...:         count = count + 1
    ...: print(count)
    ...:
3
```

Looping

```
word = 'banana'
count = 0
for letter in word:
    if letter == 'a':
        count = count + 1
        print(count)
```

Looping

```
word = 'banana'
count = 0
for letter in word:
    if letter == 'a':
        count = count + 1
    print(count)
```

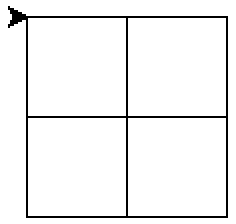


```
In [23]: word = 'banana'
    ...: count = 0
    ...: for letter in word:
    ...:     if letter == 'a':
    ...:         count = count + 1
    ...:     print(count)
    ...:
0
1
1
2
2
3
```

Loops:
Nested for Loops

put a loop inside another loop

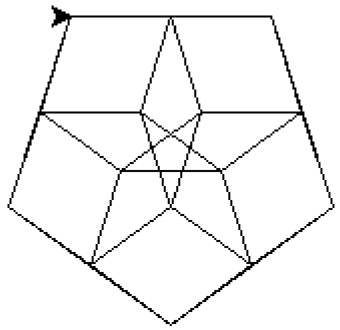
```
import turtle
for steps in range(4):
     turtle.forward(100)
     turtle.right(90)
     for moresteps in range(4):
         turtle.forward(50)
         turtle.right(90)
```



PROFESSIONAL EDUCATION TRAINING

put a loop inside another loop

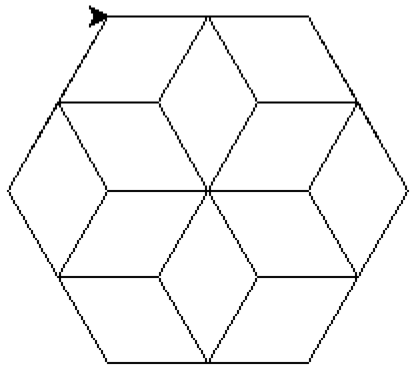
```
import turtle
for steps in range(5):
     turtle.forward(100)
     turtle.right(360/5)
     for moresteps in range(5):
          turtle.forward(50)
          turtle.right(360/5)
```



Loops:
Variables in Loops

Create Variables

```
import turtle
nbrSides = 6
for steps in range(nbrSides):
     turtle.forward(100)
     turtle.right(360/nbrSides)
     for moresteps in range(nbrSides):
          turtle.forward(50)
          turtle.right(360/nbrSides)
```



Create Variables

When we use a variable and we want to change a value that appears in many places, we only have to update one line of code!

```
import turtle
nbrSides = 6
for steps in range(nbrSides):
     turtle.forward(100)
     turtle.right(360/nbrSides)
     for moresteps in range(nbrSides):
         turtle.forward(50)
         turtle.right(360/nbrSides)
```

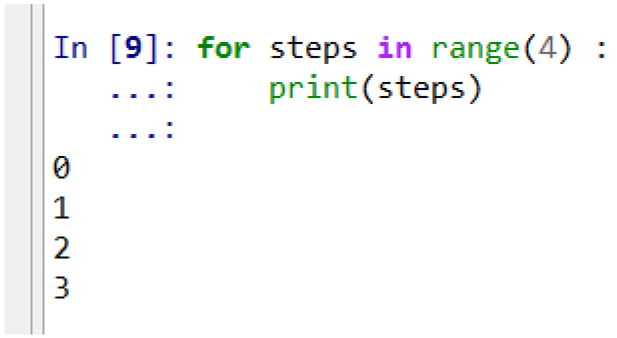
Loops:

Loop Values

Values in Loops

Yes, counting starts at zero in for loops, that’s pretty common in programming

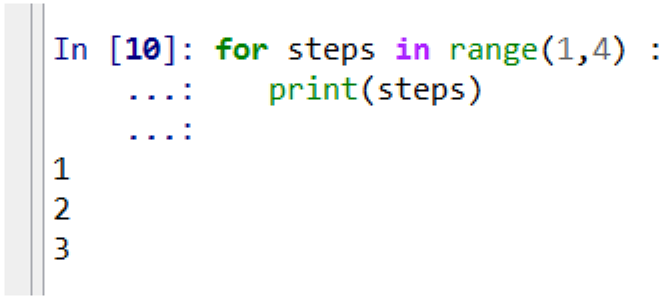
```
for steps in range(4) :
      print(steps)
```



```
In [9]: for steps in range(4) :
   ...:     print(steps)
   ...:
0
1
2
3
```

start counting from “1”

```
for steps in range(1,4) :
        print(steps)
```

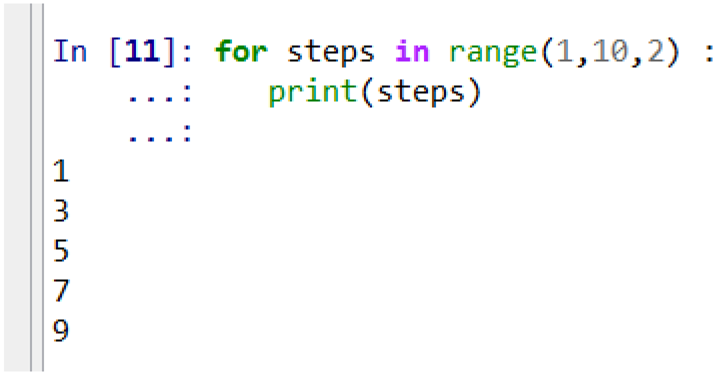


```
In [10]: for steps in range(1,4) :
    ...:     print(steps)
    ...:
1
2
3
```

PROFESSIONAL EDUCATION TRAINING

specifying a step

```
for steps in range(1,10,2) :
      print(steps)
```



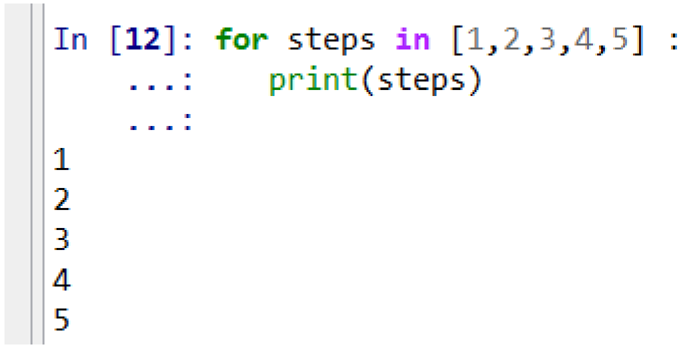
```
In [11]: for steps in range(1,10,2) :
    ...:     print(steps)
    ...:
1
3
5
7
9
```

Brackets

One of the cool things about Python is the way you can tell it exactly what values you want to use in the loop

This requires using [ ] brackets instead of ( ) and you don't use the "range" keyword

```
for steps in [1,2,3,4,5] :
      print(steps)
```



```
In [12]: for steps in [1,2,3,4,5] :
    ...:     print(steps)
    ...:
1
2
3
4
5
```

PROFESSIONAL EDUCATION TRAINING

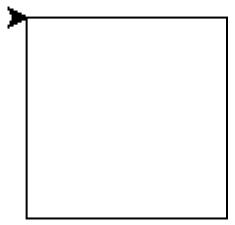
you don’t have to use numbers

```
import turtle
for steps in ['red','blue','green','black'] :
     turtle.color(steps)
     turtle.forward(100)
     turtle.right(90)
```

Loops:
`while` Loops

Draw a Square Using a while Loop

```
import turtle
counter = 0
while counter < 4:
     turtle.forward(100)
     turtle.right(90)
     counter = counter+1
```



for vs loop

Both loops have the same end result

```
import turtle
for steps in range(4):
      turtle.forward(100)
      turtle.right(90)
```

```
import turtle
counter = 0
while counter < 4:
     turtle.forward(100)
     turtle.right(90)
     counter = counter+1
```

How many lines will this loop draw?

It will actually draw 5 lines! Not 4!

```
import turtle
counter = 0
while counter <= 4:
     turtle.forward(100)
     turtle.right(90)
     counter = counter+1
```

How many lines will this loop draw?

It will draw only 3 lines! Not 4!

```
import turtle
counter = 1
while counter < 4:
     turtle.forward(100)
     turtle.right(90)
     counter = counter+1
```

How many lines will this loop draw?

Trick question! It will execute forever! Because the value of counter is never updated! How can the counter ever become greater than 3? This is called an endless loop and sometimes we get one by mistake.

```
import turtle
counter = 0
while counter < 3:
     turtle.forward(100)
     turtle.right(90)
```

Using while

Another is the while statement. Here is a version of countdown that uses a while statement

```
def countdown(n):
    while n > 0:
        print(n)
        n = n - 1
    print('Blastoff!')
```

Using while

```
def countdown(n):
    while n > 0:
        print(n)
        n = n - 1
        print('Blastoff!')
```

Using while

```
>>> fruit = 'banana'
>>> len(fruit)


index = 0
while index < len(fruit):
    letter = fruit[index]
    print(letter)
    index = index + 1
```

Loops:
Loop Control Statements

Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Python supports the following control statements. Click the following links to check their detail.

Function	Description
break	Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue	the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
pass	**The** pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

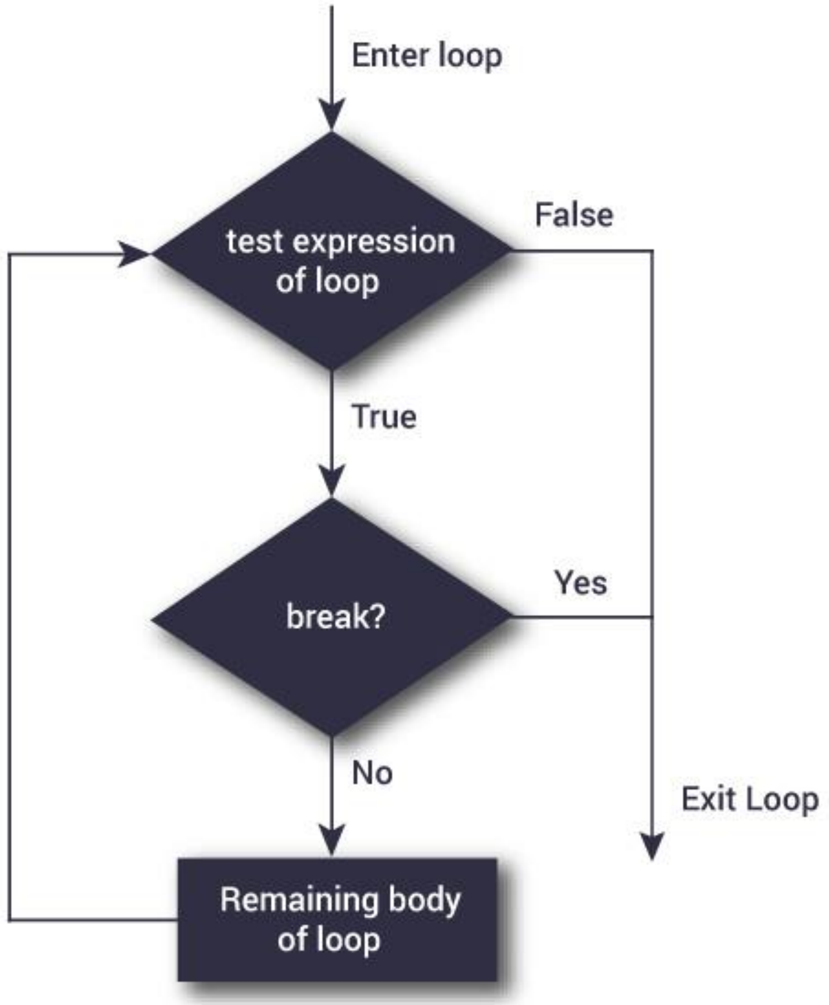
break statement

It terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C.

The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The break statement can be used in both while and for loops.

If you are using nested loops, the break statement stops the execution of the innermost loop and start executing the next line of code after the block.

Break


Enter loop
test expression of loop
False
True
break?
Yes
No
Exit Loop
Remaining body of loop

break

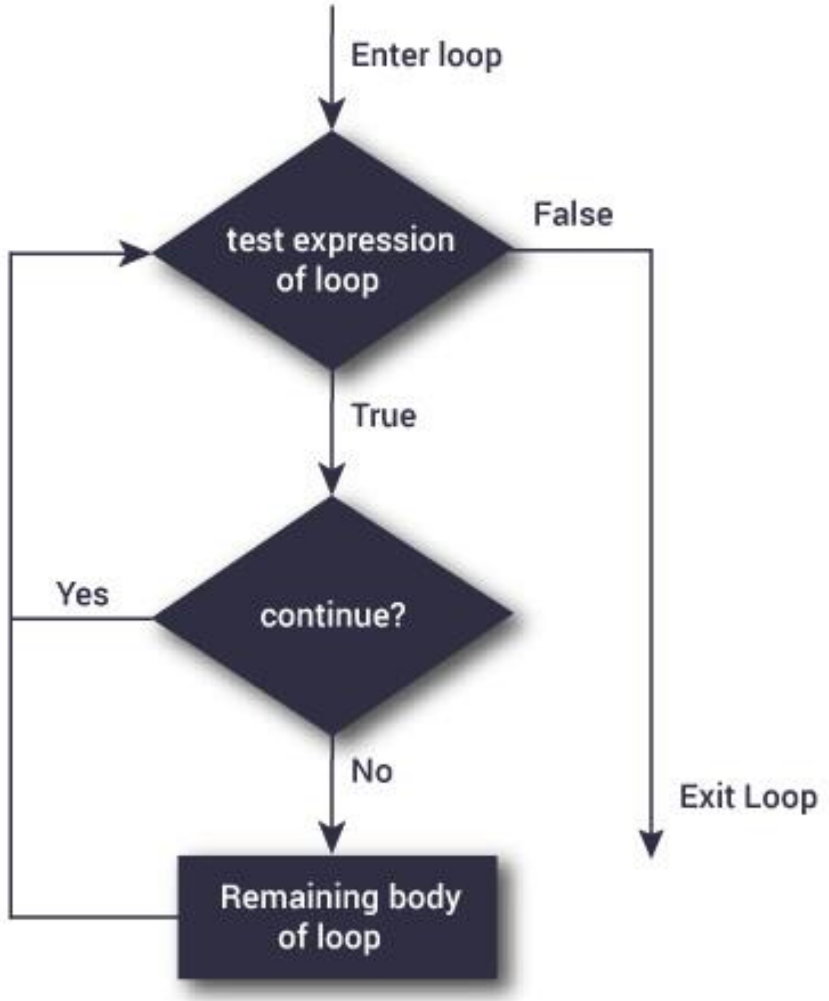
```
for letter in 'Python':     # First Example
   if letter == 'h':
      break
   print 'Current Letter :', letter
var = 10                    # Second Example
while var > 0:
   print 'Current variable value :', var
   var = var -1
   if var == 5:
      break
print "Good bye!"
```

Continue Statement

It returns the control to the beginning of the while loop.. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop.

The continue statement can be used in both while and for loops.

continue


Enter loop
test expression of loop
False
True
Yes
continue?
No
Exit Loop
Remaining body of loop

continue

```
for letter in 'Python':      # First Example
   if letter == 'h':
      continue
   print 'Current Letter :', letter
var = 10                     # Second Example
while var > 0:
   var = var -1
   if var == 5:
      continue
   print 'Current variable value :', var
print "Good bye!"
```

Case Studies:
Factorial

Calculate Factorials

The factorial of a number is the product of all the integers from 1 to that number.

For example, the factorial of 6 (denoted as 6!) is 1*2*3*4*5*6 = 720. Factorial is not defined for negative numbers and the factorial of zero is one, 0! = 1.

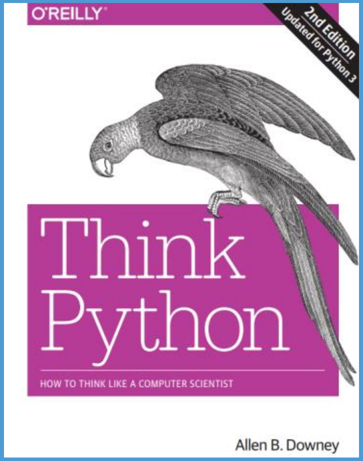
Calculate Factorials

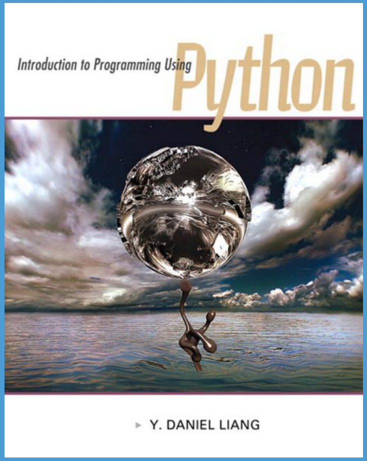
```
# Python program to find the factorial of a number
provided by the user.

# change the value for a different result

num = 7

# uncomment to take input from the user

#num = int(input("Enter a number: "))

factorial = 1

# check if the number is negative, positive or zero

if num < 0:

   print("Sorry, factorial does not exist for negative
numbers")

# to be cont'd
```

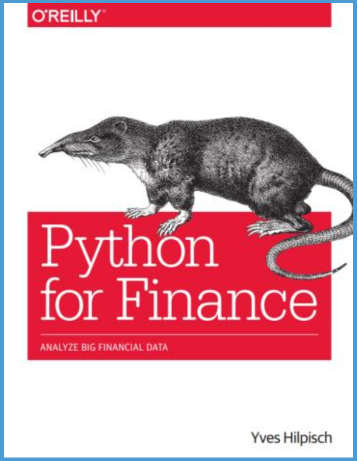
Calculate Factorials

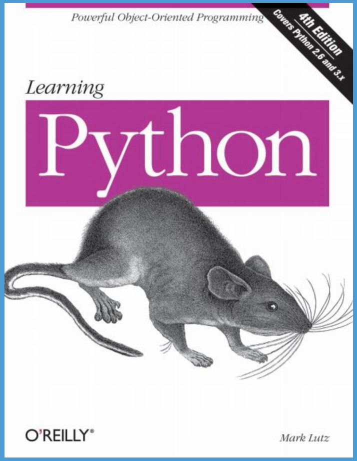
```
elif num == 0:
   print("The factorial of 0 is 1")
else:
   for i in range(1,num + 1):
       factorial = factorial*i
   print("The factorial of",num,"is",factorial)
```

References


O'REILLY
2nd Edition
Updated for Python 3
Think Python
HOW TO THINK LIKE A COMPUTER SCIENTIST
Allen B. Downey


Introduction to Programming Using Python
Y. DANIEL LIANG


O'REILLY
Python for Finance
ANALYZE BIG FINANCIAL DATA
Yves Hilpisch


Powerful Object-Oriented Programming
4th Edition
Covers Python 2.6 and 3.x
Learning Python
O'REILLY
Mark Lutz