**Practical Programming Using Python**

Contents

[# Formatted printing (p.36) 2](#_Toc363486056)

[2-1: The Guess and Check Pattern 2](#_Toc363486057)

[# 04\_Compare two floating point numbers.py 3](#_Toc363486058)

[# 07\_Nested For Loops.py -- Kent D. Lee 4](#_Toc363486059)

[# 3-3: Guess and Check for Lists -- Kent D. Lee 4](#_Toc363486060)

[# 3.4 - The Accumulator Pattern for Lists -- Kent D. Lee 5](#_Toc363486061)

[# 4-1: Creating and Using Objects -- Kent D. Lee 5](#_Toc363486062)

[# 4-3: Dictionaries in Python -- Kent D.Lee 6](#_Toc363486063)

[# 5-1: Defining and Calling Functions -- Kent D.Lee 7](#_Toc363486064)

[5-3: The Run-time Stack 8](#_Toc363486065)

[# 5-5: The Main Function -- Kent D. Lee 8](#_Toc363486066)

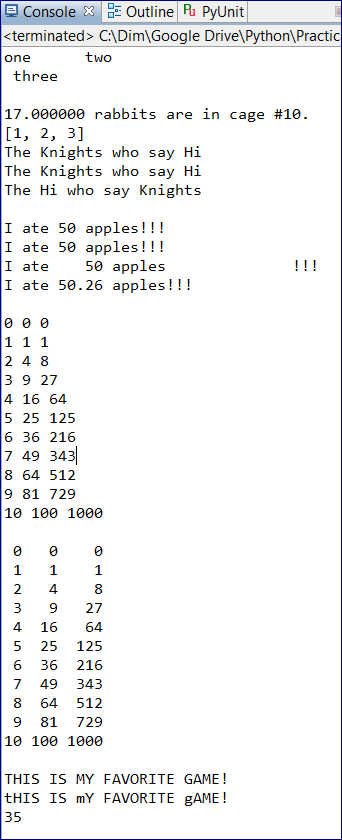
[# 7-1: Object-Oriented Programming 9](#_Toc363486067)

[# 7-2: Inheritance in Python 12](#_Toc363486068)

#===============================================================================

# # Formatted printing (p.36)

#===============================================================================



rabbits = 17

cage = 10

**print**("%f rabbits are in cage #%d. " % (rabbits,cage))

**print** ("[%i, %i, %i]" % (1, 2, 3))

**print**("The {} who say {}".format('Knights','Hi'))

**print**("The {0} who say {1}".format('Knights','Hi'))

**print**("The {1} who say {0} \n".format('Knights','Hi'))

**print**("I ate {} {}!!!".format(50,'apples'))

**print**("I ate {0} {1}!!!".format(50,'apples')) # {0} takes 50 and {1} takes 'apples'

**print**("I ate {0:5} {1:20}!!!".format(50,'apples')) # add extra spaces after the argument

**print**("I ate {0:5.2f} {1}!!! \n".format(50.259,'apples')) # round to 2 decimal points

# print("".format()) -- general format

**for** i **in** range(11):

**print**("{0} {1} {2}".format(i,i\*i,i\*i\*i))

**print**()

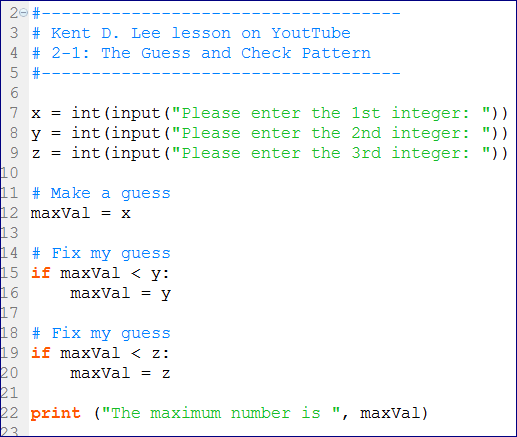
**for** i **in** range(11):

**print**("{0:2d} {1:3d} {2:4d}".format(i,i\*i,i\*i\*i))

# 2-1: The Guess and Check Pattern

If we to get, e.g., the BIGGEST number.

* We assign maxVal = any number
* Compare with thenext number
* Fix the maxVal if our guess was wrong



#===============================================================================

# # 04\_Compare two floating point numbers.py

#===============================================================================

#------------------------------------------------------------------------------

# Compare two floating point numbers for equality

# as the == won't work in any language

#------------------------------------------------------------------------------

# Real numbers, e.g. 3.14... are infinite numbers

# and floating point numbers (in Python) are approximation of the real numbers

# basically Python rounds up those numbers at some point

# Therefore we cannot directly compare two floating point numbers as

# 6.33 = 6.33 (because 6.333333331 is not equal to 6.3333333337)

**import** random

x = random.random()

xVal = int(x\*10000)/100.0

y = random.random()

yVal = int(y\*10000)/100.0

answer = float(input("What is the value of " + str(xVal) + "-" + str(yVal) + " equal to? "))

realAnswer = xVal - yVal

# When you compare two floats

# 1) Substract one float from the other

# 2) Take an absolute value of the substraction

# 3) Devide by the real answer (to get the deviation from the real answer)

# 4) Compare for enaquality with some possible error, e.g. 0.01

**if** abs((answer - realAnswer)/realAnswer) < 0.001:

**print** ("You did tit!!!")

**else**:

**print**("No, the real answer was", realAnswer)

#===============================================================================

# # 07\_Nested For Loops.py -- Kent D. Lee

#===============================================================================

**print**(" ",end="")

**for** i **in** range(16):

**print**("%5d|" %i, end='')

**print**()

**print**("------"\*17)

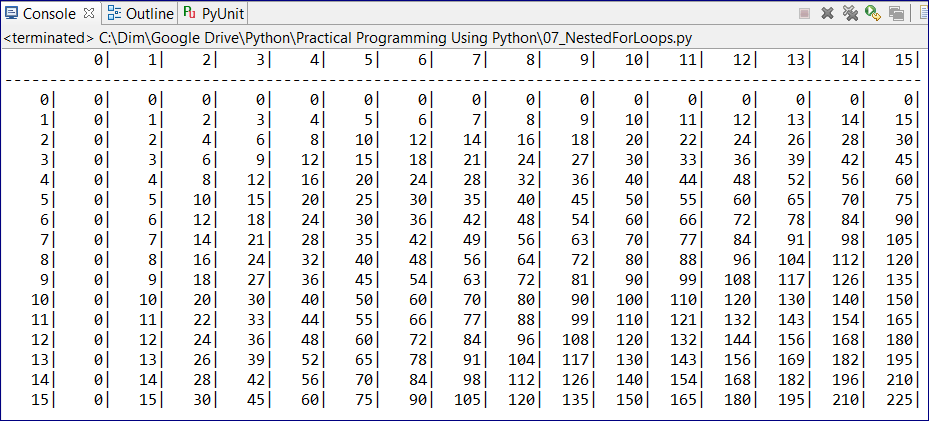
**for** i **in** range(16):

**print**("%5d|" %i,end="")

**for** j **in** range(16):

**print**("%5d|" %(i\*j),end="")

**print**()



#===============================================================================

# # 3-3: Guess and Check for Lists -- Kent D. Lee

#===============================================================================

# Check if a number between 2 and 49 is a PRIME number

primes = [2,3,5,7]

**print**(primes)

x = int(input('Please enter an integer between 2 and 49: '))

# Make a guess

isPrime = **True**

# Go through the list to check if our guess is wrong

**for** p **in** primes:

# check if there is no remainder then

# the number x s the prime

**if** x % p == 0 **and** **not** x **in** primes:

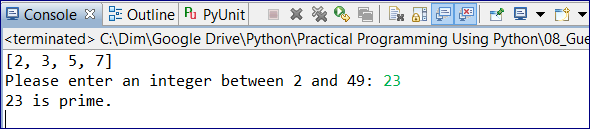
isPrime = **False**

**if** isPrime:

**print**(x, "is prime.")

**else**:

**print**(x, "is NOT prime.")



#===============================================================================

# # 3.4 - The Accumulator Pattern for Lists -- Kent D. Lee

#===============================================================================

#Checking sentence --> 'How are you doing? Today is a fine day for a bycicle ride.'

s = input("Please enter a sentence: ")

lst = s.split()

# Initialize our accumulators

count = 0

wordLenghts = 0

**for** word **in** lst:

count +=1

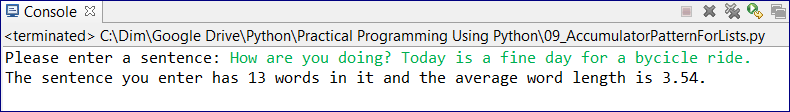
wordLenghts += len(word)

avgWordLength = wordLenghts / count

**print**('The sentence you enter has'

, count

, 'words in it and the average word length is %1.2f.' % avgWordLength)



#===============================================================================

# # 4-1: Creating and Using Objects -- Kent D. Lee

#===============================================================================

**import** turtle

# Calling the Turtle constructor to create a Turlte Object

# Constructing a "Turtle" type using imported module "turtle"

# It's similar to creating a integer x = int('6')

t = turtle.Turtle()

# "t" is a Turtle object now

# we can print --> help('turtle') to get the documentation

**t.color("#7D7EC0") # assigning a color to the line**

# Mutator methods -- change the object "t"

**for** i **in** range(5):

t.forward(200)

t.left(144)

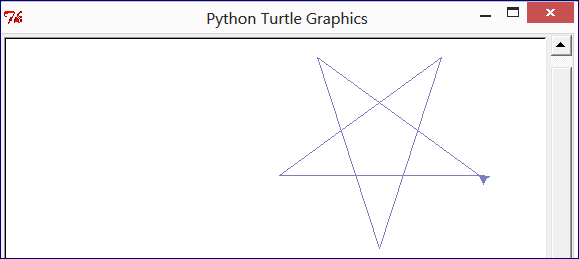
t.forward(200)

t.left(144)

# getscreen() is an accessor method on Turtle object

screen = t.getscreen()

screen.exitonclick()



#===============================================================================

# # 4-3: Dictionaries in Python -- Kent D.Lee

#===============================================================================

#-----------------------------------------

# This is a list

#-----------------------------------------

myList = [] # empty list

lst = ['Kent','Sophus','Lee']

**print**(lst[0])

**for** name **in** lst:

**print**(name)

**print**("-------------------------------------------")

#-----------------------------------------

# This is a dictionary

#-----------------------------------------

# A dictionary has a "Key" and value(s) associated with the key

# Keys have to be UNIQUE

dictionary = {} # An empty dictionary

dictionary['Kent'] = 0

dictionary['Sophus'] = 1

dictionary['Lee'] = [4,6,7]

**print**(dictionary)

# there is no order in a dictionary when printing keys

**for** key **in** dictionary:

**print**(key)

# printing keys and values

**for** key **in** dictionary:

**print**(key, dictionary[key])

**print**("-------------------------------------------")

pets = {}

pets['Kent'] = 'Mesa'

pets['Sophus'] = 'Smudge'

pets['Lee'] = 'Lassie'

**for** owner **in** pets:

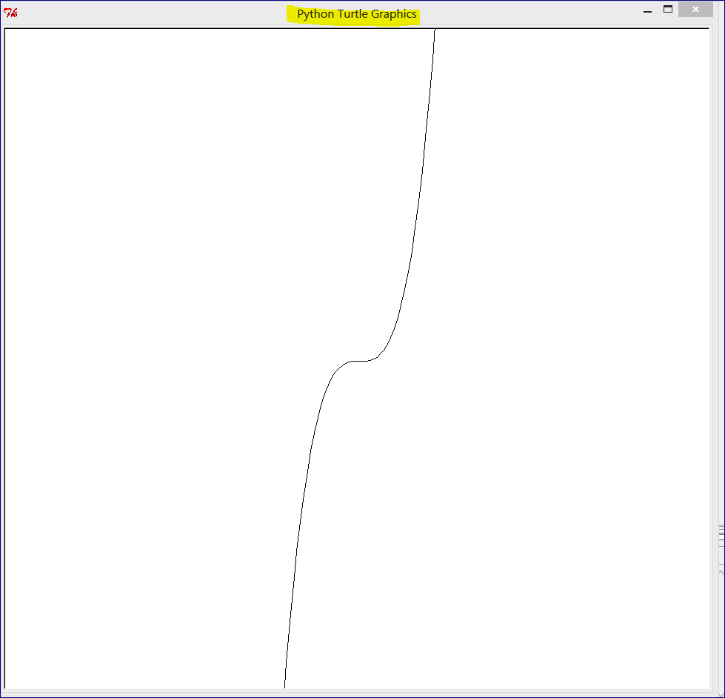
**print**(owner, pets[owner])

#===============================================================================

# # 5-1: Defining and Calling Functions -- Kent D.Lee

#===============================================================================

# define a function doesn't mean it's execution

**import** turtle

**def** f(x):

**return** x\*x\*x # cube of x

t = turtle.Turtle()

screen = t.getscreen()

screen.setworldcoordinates(-10,-10,10,10)

t.penup()

t.goto(-20, -20)

t.pendown()

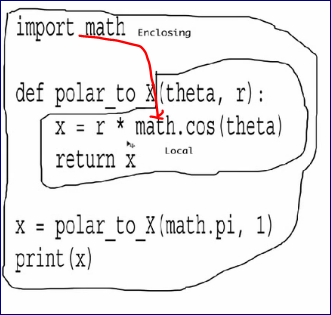
**for** k **in** range(-30, 30, 1):

myX = k/10

myY = f(myX)

t.goto(myX, myY)

screen.exitonclick()



# 5-3: The Run-time Stack

Contains a stack of activation records

#===============================================================================

# # 5-5: The Main Function -- Kent D. Lee

#===============================================================================

The **\_\_name\_\_** variable is created by Python and set to the value "\_\_main\_\_" if the module being executed is the main module. If the module was imported into﻿ another module, then \_\_name\_\_ would not be set to "\_\_main\_\_". This mechanism is in place so that each module can have its own main function (for testing purposes) and only the main function from the main module will be executed.

**def** add(a,b):

c = a+b

**return** c

**def** mult(a,b):

d = a\*b

**return** d

# Define a main() function that will run the whole code. It's not mandatory in Python as it's in other programming languages

# e.g. in Java we write Public Static Void Main function to get things started

**def** main():

a = 2

b = 7

**print**(add(a,b))

**print**(mult(a,b))

# It will execute the main program

**if** \_\_name\_\_ == '\_\_main\_\_':

main()

# Create main() function does two things:

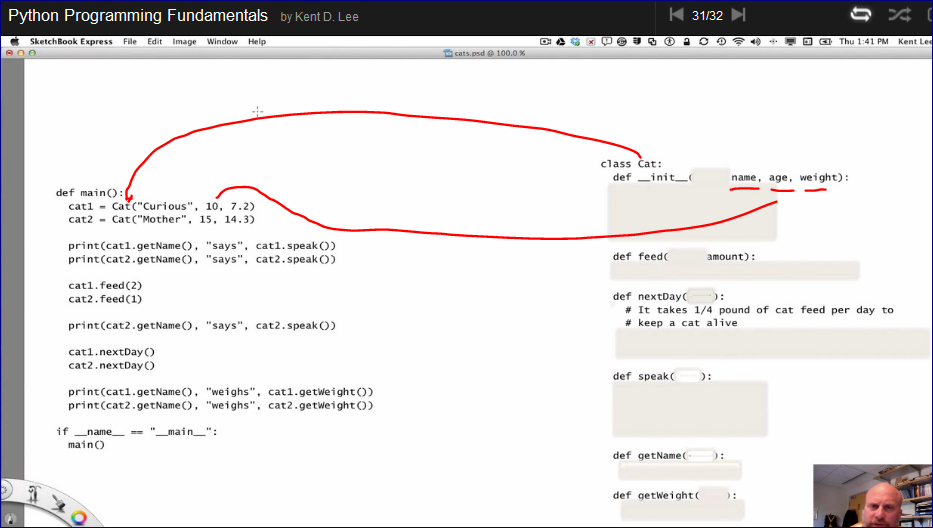
# 1) It hides variables that are declared in the mainfunction from the rest of the program. That makes the code more secure

# 2) If a module is imported to that program it will not have '\_\_main\_\_' name and won't be executed

#===============================================================================

# # 7-1: Object-Oriented Programming

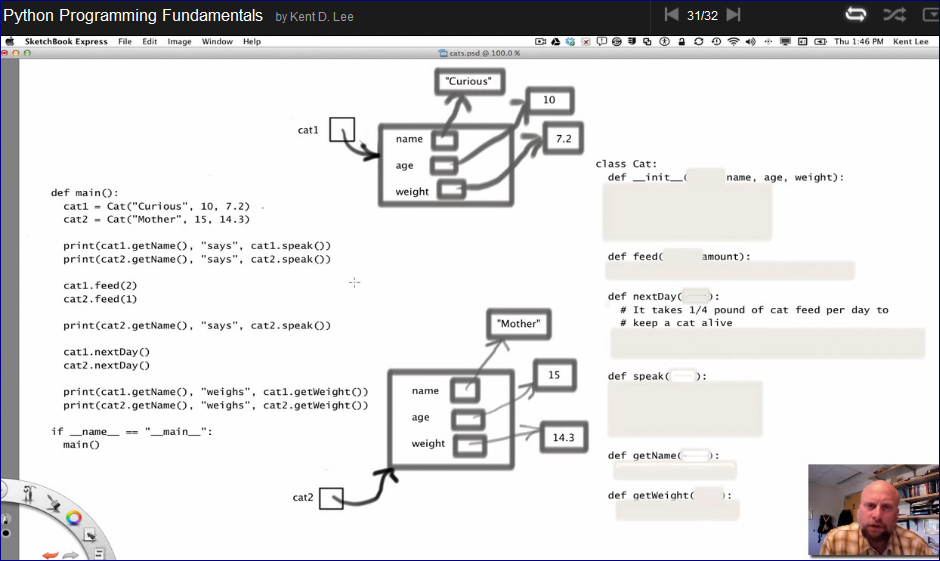
#===============================================================================

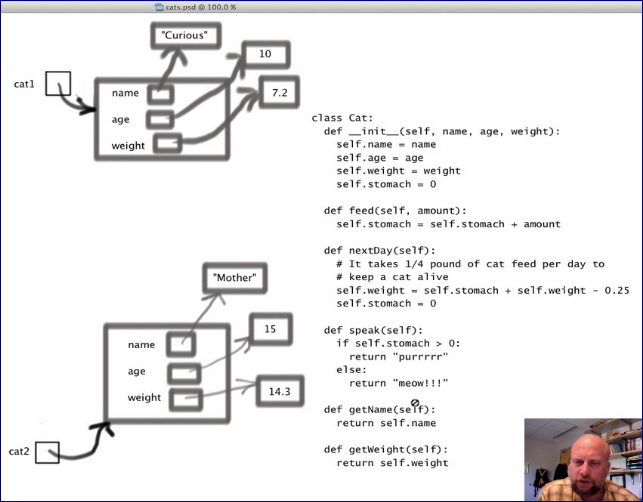


This is how we see objects and classes.

We can provide name, age, and weight to create an object.

Then we can use different methods such as feed, nextDay, speak, getName, and getWeight on this object.





#===============================================================================

# 7-1: Object-Oriented Programming

#===============================================================================

# If I want to describve a "cat" object I need to write (declare) a "cat" class first

# The First part of the class is called a "Constructor" and it has a special name \_\_init\_\_

# In that constructor we declare attributes that we want to provide when declare a cat object

# Inside of the class we create methods -- actions that may be used on the class's objects

#---------------------------------------------

**# Defining a class and methods of the class**

#---------------------------------------------

# 'self' parameter is an object itself

# 'self' is a special pointer to the object you are working with right now

**class** Cat:

**def** \_\_init\_\_ (**self**, name, age, weight):

**self**.name = name

**self**.age = age

**self**.weight = weight

**self**.stomach = 0

**def** feed(**self**, amount):

**self**.stomach = **self**.stomach + amount

**def** nextDay(**self**):

# it takes of 1/4 pound of cat feed per day to keep cat alive

**self**.weight = **self**.stomach + **self**.weight - 0.25

**self**.stomach = 0

**def** speak(**self**):

**if** **self**.stomach > 0:

**return**('purrrrr')

**else**:

**return**('meow!!!')

# Create a getName method. This method may be used on a "cat" object

**def** getName(**self**):

**return** **self**.name

**def** getWeight(**self**):

**return** **self**.weight

#-------------------------------------------

**def** main():

cat1 = Cat("Curious", 10, 7.2)

cat2 = Cat('Mother', 15, 14.3)

**print**(cat1.getName(),'says',cat1.speak())

**print**(cat2.getName(),'says',cat2.speak())

cat1.feed(2)

cat2.feed(1)

**print**(cat2.getName(),'says',cat2.speak())

cat1.nextDay()

cat2.nextDay()

**print**(cat1.getName(),'weighs',cat1.getWeight())

**print**(cat2.getName(),'weighs',cat2.getWeight())

#-------------------------------------------

**if** \_\_name\_\_ == '\_\_main\_\_':

main()

#===============================================================================

# # 7-2: Inheritance in Python

#===============================================================================

**# Inheritance allows to use the same class but with some changes in it**

**class** Cat:

**def** \_\_init\_\_ (**self**, name, age, weight):

**self**.name = name

**self**.age = age

**self**.weight = weight

**self**.stomach = 0

**def** feed(**self**, amount):

**self**.stomach = **self**.stomach + amount

**def** nextDay(**self**):

# it takes of 1/4 pound of cat feed per day to keep cat alive

**self**.weight = **self**.stomach + **self**.weight - 0.25

**self**.stomach = 0

**def** speak(**self**):

**if** **self**.stomach > 0:

**return**('purrrrr')

**else**:

**return**('meow!!!')

# Create a getName method. This method may be used on a "cat" object

**def** getName(**self**):

**return** **self**.name

**def** getWeight(**self**):

**return** **self**.weight

#++++++++++++++++++++++++++++++++++++++++++++++

**class** bigCat(Cat):

**def** \_\_init\_\_(**self**, name, age, weight):

super().\_\_init\_\_(name, age, weight)

**def** speak(**self**):

**return** 'This a big cat'

#++++++++++++++++++++++++++++++++++++++++++++++

#-------------------------------------------

**def** main():

cat1 = Cat("Curious", 10, 7.2)

cat2 = Cat('Mother', 15, 14.3)

**print**(cat1.getName(),'says',cat1.speak())

**print**(cat2.getName(),'says',cat2.speak())

cat1.feed(2)

cat2.feed(1)

**print**(cat2.getName(),'says',cat2.speak())

cat1.nextDay()

cat2.nextDay()

**print**(cat1.getName(),'weighs',cat1.getWeight())

**print**(cat2.getName(),'weighs',cat2.getWeight())

**print**(cat1, cat2)

#++++++++++++++++++++++++++++++++++++++++++++++++

cat3 = bigCat("Chebik", 12, 10) # cat3 is an object of the new class: "bigCat"

**print**(cat3)

**print**("My name is " + str(cat3.getName()) + ".") # getName() method is the same as it's for Cat object

**print**(cat3.speak()) # Speak() method is different. It's rewritten in the new class "bigCat"

#++++++++++++++++++++++++++++++++++++++++++++++++

#-------------------------------------------

**if** \_\_name\_\_ == '\_\_main\_\_':

main()

