V. Rego

Sept. 15, 2015

Week 4, Examples 1

```python
#1.py

def wait():
    x = input(" ")

# We'll introduce strings, show they are immutable, show how to access
# each character in a string, and show how to print

# First, a simple function to say Hello.

def Hi(s):
    # remember s is a string that is given to Hi as input
    print("Hello there," , s , "!")

def HI(s):
    # remember s is a string that is given to Hi as input
    print("Hello there," , s , "!", sep="")

# When you define

wait()  
s = "Joe"  
# you can use single quotes or double quotes
# C uses double quotes

wait()  
s  
# Python prompt responds with 'Joe' (it likes to use
# single quotes) to acknowledge this string definition

wait()  
print(s)  
# but here it just prints Joe; it assumes you know, because
# print works with strings

wait()  
print(s[2]," ",s[1]," ",s[0])  
# printing what is in s, element by element, backwards

# That string s is now immutable (cannot be changed). Say you want to
# change it to "Moe". Assigning s[0] = "M" will cause a trace-back
```
#s[0] = "M"  # we'll comment it out, else program will bomb here

# Suppose Cedric (Captain of the crew-cut society) is new in town and wishes
# for cs177. He runs into you in the cafeteria. You call your "Hello"
# function

wait()
Hi("Crewcut Cedric!")  # notice print leaves a space between print items

wait()
HI("Crewcut Cedric")  # sep="" suppresses this white-space separator

print("\n")  # blank line

# a loop to print the name one character at a time

wait()
x = "Crewcut Cedric"

for j in range(0,len(x),1):
    print(x[j],end="")

# the "c" at the end is accessible by index -1, the "i" by -2, etc

wait()
print("\n\n")  # two blank lines

# You can use "negative indexing" reach elements in x. Say you want to
# print the string in x backwards

for j in range(-1,-1*(len(x)+1),-1):  # remember s[-1] is the last char "r"
    print(x[j],end="")

#So now you know a way to print backwards; of course, you could have done
#it without the negative indices, by starting at len(x)-1 and decrementing
#the index by one for each character going backwards

#
#2.py

def wait():
    x = input(" ")

# A little more of the same, to get you comfy with strings and loops and
# printing
# What is a string?
# A string is a sequence of 0 or more characters

# What should you know when working with a string?
# Length of the string, position of each char in the string, and that
# a string is *immutable* (you cannot change its characters)

# Chars are numbered 0,1,2,..... len(str)-1
# where element 0 is the posn of the first char
# len(str)-1 is the posn of the last char, and
# len(str) is a function that returns the length of the string

def main():
    
    s = "I'm Bilbo Baggins!"
    wait()
    print("s :", s)
    print(""

    #Note: Python allows you to use either " or ' (consistently) to define
    strings.

    #But suppose I use '. What happens when I have a grammatical ' in string, as
    # above ?

    # s1 = 'I'm Bilbo Baggins'     ===> syntax error, but try "escaping" with \

    wait()
    s1 = 'I\'m Bilbo Baggins!'   # the \ tells python: this is a real '
    print("s1:", s1)
    print(" ")

    wait()
    print("How are these characters stored in string s? \n")

    #Let's print the positions of chars in s

    for j in range(0,len(s),1): #positions
        print(j," ",end="")

    print("\n")
    for j in range(0,len(s),1):  #char in that position

        if (j < 10):
            print(s[j]," ",end="")  #no problem, one char under each digit
        else:
            print(s[j]," ",end="")
# Try to change any character and it won't work for a string
# Example: s[11]= "L", say you want the last name "Laggins"
# You'll get an error and trace-back

# So once defined, strings are "constants" or "immutable"

#

#3.py

def wait():
    x = input(" ")

# How does the computer store strings?

# This will tell you how each character is stored in a byte (8 bits)
# using the ASCII code. We ask for the character code (integer) and
# walk up the bits (from rightmost to leftmost) checking if it's a 1
# or a 0. We do this by right-shifting the computer word containing
# the byte

# We will only have to work with *positive* integers

# (negative integers are changed a tiny bit before they are stored) because
# we have to encode the "-" sign too, in some way

def main():
    wait()
    ch = input("Type a single character: ")
    print("You typed ",ch)

    wait()
    print("But this character is stored as the integer ",ord(ch))
    print(" encoded in binary, because that is it\'s ASCII code \n")

    wait()
    print("Now 8-bits (in truth, 7 bits) suffice to hold")
    print("each character in ASCII. But what would these 8 bits,\")
    print("also called a byte, look like?\")

    wait()

# 1. Ask func. ord to tell us what the integer code for the char. is. Then
# 2. in a loop, check the last bit to see if it's a 0 or a 1, and then
# 3. shift the entire word (32- or 64-bits, depending on processor) to
# 4. the right by one bit, and repeat. Only need to do this 8 times since
#    we are only interested in the lowest (rightmost) byte in the word.
n = ord(ch)  # "byte" is an empty list originally, but we'll
# append one bit to it at a time

for i in range(0, 8):
    if((n%2) == 0):
        bit = 0
    else:
        bit = 1

    byte.append(bit)  # adding to the list from the right, and so
# unfortunately, we'll get the byte in reverse

    n = n >> 1

# But it's not a big problem since a simple loop will reverse it

rbyte = []  # list "rbyte" is the reverse of list "byte"
for j in range(7, -1, -1):
    rbyte.append(byte[j])  # bit-8 -> position 1, bit-7 -> position 2, etc.

# Now print rbyte[]. Remember it's a list and will contain commas.
# We can't use n since its been right-shifted away. Call "ord" again

wait()
print("\n Decimal ",ord(ch)," appears as ",rbyte," in binary (inside a byte")

# Because we want to see the actual byte (without commas), let's print
# using "print" in a loop, but all output on same line

wait()
print("\n The decimal number ",ord(ch), "in binary is: ",end="")
for i in range(0, 8):
    print(rbyte[i],end="")

# The print func. accepts an end parameter which defaults to "\n".
# Setting it to an empty string prevents it from issuing a new line
# at the end of the line.

# The program is done.

# Homework: Write a simple loop that takes the binary number in a byte as
# input and returns the decimal number it represents, i.e., ord(ch)
def wait():
    x = input(" ")

# Now that you understand the code in example 3.py, lets see how any name
# is stored. It will look like a long string of bytes in memory

def getasciicode(c):
    n = ord(c)
    byte = []

    # We used two lists and two loops earlier. We'll use a shortcut now.
    for i in range(0, 8):
        if((n%2) == 0):
            bit = 0
        else:
            bit = 1

        byte.append(bit)  # adding to the list from the right, and so
        # unfortunately, we'll get the byte in reverse

        n = n >> 1

    #invoke helpful function to reverse "in place" [no extra list like rbyte is
    # required]
    byte.reverse()
    return(byte)  # and return it

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

def printbyte(list):
    for j in range(0, 8):
        print(list[j],end="")
    print(" ",end="")

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

def main():

    name = input("Type in any name: ")

    #The byte-strings will be contiguous inside a computer word, but we'll use
    #spaces so you can see each byte separately

    wait()
    for i in range(0,len(name)):
        code = getasciicode(name[i])
        printbyte(code)  #removes the list commas and prints byte
#5.py

def wait():
    x = input(" ")

#You know plenty about how to *ITERATE through characters* in a string

#Now let's look at SLICING, REPETITION and CONCATENATION

def main():

    s = "Moody Rudy"

    print(" ")
    print("First look at SLICES\n")
    print(s)
    print(" ")

    wait()
    print("Slice s[0:4] is ",s[0:4]) # like range, excludes the 4th

    wait()
    print("Slices s[1:5] and s[7:10] give ",s[1:5],s[7:10],sep="") #removed space

    wait()
    print("Slice s[:5] is",s[:5])
    wait()
    print("Slice s[5:] is",s[5:])
    wait()
    print("Slice s[:] is",s[:])

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

    wait()
    print(" ")
    print("Next look at CONCATENATION")
    print(" ")
    wait()
    print("sugar" + "and" + "jam" + "and" + "pickles" + "and" + "ham")

    wait()
    m = "Moody"
    s = " 
    r = "Rudy"
    l = "loves"
    b = "broody"
t = "Trudy"

z = m + s + r + s + l + s + b + s + t  # we've use s for space

print(" ")
print(z)
print(" ")

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

# Repetition is what may happen as Crewcut Cedric sees a
cool, long-haired CS177 hippie strolling across campus

# It makes him want to scream:

wait()
print("Cedric demonstrates the CONCATENATION of strings")
print(" ")

s = "Cut his hair!"

n = 3

wait()
print(n*s)

print(" ")

wait()
print("and if screaming ",n," times won't do .... ")

print(" ")

n = 100
wait()
print(n*s)

wait()
print(" ")
print(10*"Cut!" + 5*"Hair")

#

#6.py

# Combining what we've learned from (a) graphics and (b) lists
# we'll make a table of "Matilda's Weightloss Plan", i.e., the
# food-group she plans to focus on the most in each month. It may
# be a radical plan, but we're only interested in the table! :)

http://courses.cs.purdue.edu/
# Note: We are keeping month and food-group information in STRINGS
# and *indexing* into these strings to pick off the right month and
# food-group

```python
from graphics import *

def wait():
    x = input(" ")

def box(m, cstring, botleftx, botlefty, w):
    s = Rectangle(Point(botleftx, botlefty), Point(botleftx+12, botlefty+1))
    p = (12-m)*1
    code = cstring[p:p+1]
    c = "blue"
    if (code == "R"): # We'll see a better way to
        c = "red"
    if (code == "G"): # check for different possibilities
        c = "green"
    if (code == "Y"): # later
        c = "yellow"
    s.draw(w) # we're going to draw the box, and then
    wait()
    s.setFill(c) # fill it with the right color
    wait()
    lin = Line(Point(botleftx+5,botlefty), Point(botleftx+5,botlefty+1))
    lin.draw(w)

def month(n, mstring, botleftx, w):
    p = (n-1)*3
    t = Text(Point(botleftx+3, 13-n+0.5), mstring[p:p+3])
    t.setSize(18)
    t.setStyle("bold")
    t.draw(w)

def food(n, fstring, botleftx, w):
    p = (n-1)*5
    t = Text(Point(botleftx+8, 13-n+0.5), fstring[p:p+5])
    t.setSize(18)
    t.setStyle("bold")
    t.draw(w)
```

Computer Science Courses - http://courses.cs.purdue.edu/
def main():

    w = GraphWin("Waltzing Matilda's Weightloss Plan",400,800)
    w.setCoords(0,0,14,14)

    r = Rectangle(Point(1,1),Point(13,13))
    r.setWidth(3)
    r.draw(w)

    monthstring = "JanFebMarAprMayJunJulAugSepOctNovDec"
    foodstring = "CarroBeansBroccBeetsSpinaPotatChardAsparCauliYamsTurniCakes"
    colorstring = "RGGRGYGGYRYB"

    for j in range(1,13,1):
        wait()
        box(j,colorstring,1,1+j-1,w)
        wait()
        month(13-j,monthstring,1,w)
        wait()
        food(13-j,foodstring,1,w)

#__________________________________________________________________________
#7.py
# This example is the same as the previous (Matilda, 6.py) but
# we'll use LISTS instead of strings

# A Python LIST is also a "sequence",like a string, but it is more
# general and flexible than a string. It is mutable, i.e., you can
# change its elements. You can also apply all the string operations
# to a list. Just like you'll learn string methods (yes, strings are
# objects) you'll also learn list methods, because lists are also objects.

# Note: We are keeping month and foodgroup information in LISTS.
# First advantage -> we don't have to limit item sizes because of the second
# advantage
# Second advantage -> we can index into a list without slicing

# You'll notice that nearly all the code remains the same except for
# month and foodgroup information (now LISTS), and how we index into
# this information to get what we want. We've left the color information
# in a string (we could have changed that too, but since the indexing there
# was like a list index, we left it unchanged)
from graphics import *

def wait():
    x = input(" ")

def box(m, cstring, botleftx, botlefty, w):
    s = Rectangle(Point(botleftx, botlefty), Point(botleftx+12, botlefty+1))

    p = (12-m)*1
    code = cstring[p:p+1]  # still slicing thru color string in this example

    c = "blue"
    if (code == "R"):  # We'll see a better way to
        c = "red"
    if (code == "G"):  # check for diferent possibilities
        c = "green"
    if (code == "Y"):  # later
        c = "yellow"

    s.draw(w)  # we're going to draw the box, and then
    wait()
    s.setFill(c)  # fill it with the right color
    wait()

    lin = Line(Point(botleftx+5, botlefty), Point(botleftx+5, botlefty+1))
    lin.draw(w)

def month(n, mlist, botleftx, w):

    t = Text(Point(botleftx+3, 13-n+0.5), mlist[n-1])
    t.setSize(18)
    t.setStyle("bold")
    t.draw(w)


def food(n, flist, botleftx, w):

    t = Text(Point(botleftx+8, 13-n+0.5), flist[n-1])
    t.setSize(18)
    t.setStyle("bold")
    t.draw(w)

def main():
w = GraphWin("Waltzing Matilda's Weightloss Plan", 400, 800)

w.setCoords(0,0,14,14)

r = Rectangle(Point(1,1),Point(13,13))
r.setWidth(3)
r.draw(w)

monthlist = ["January", "February", "March", "April",
            "May", "June", "July", "August",
            "September", "October", "November", "December"]

foodlist = ["Carrots", "Beans", "Broccoli", "Beets",
            "Spinach", "Potatoes", "Chard", "Asparagus",
            "Cauliflower", "Yams", "Turnips", "Cakes"]

colorstring = "RGGRGYGGYRYB"

for j in range(1,13,1):
   wait()
   box(j,colorstring,1,1+j-1,w)
   wait()
   month(13-j,monthlist,1,w)
   wait()
   food(13-j,foodlist,1,w)

#_________________________________________________________________________

#8.py

# Larry is in comic prison (a prison for people who tell bad jokes). Moe
# is also in comic prison, in a different cell. A warden (Shemp) who hates
# bad jokes is keeping an eye on both of them. Larry has hatched an escape
# plan and wants to send a secret message to Moe. He does not want Shemp
# to understand the message. So we need a secret "encoding".

# We'll first use a small example with 6 characters from {A,B,C,D,E," "}

# Suppose he wants to send the string "A CAB"

# One type of encoding uses a dictionary. Let's assume that before they were
# caught, they both memorized a secret dictionary. That is:

# A   is coded as D
# B   is coded as E
# C   is coded as A
# D   is coded as B
# E is coded as C
# " " is coded as F

# So the message that Larry sends will be "DFADE", and Shemp will have no clue what this means unless he knows or can somehow get the dictionary.

# What if we make such a dictionary using A-Z and " "?

import sys

def search(list, item):
    # Look for the index of Larry's char
    for i in range(0, len(list), 1):
        if item == list[i]:
            return(i)

    print("Error: Larry's character is not in the given alphabet")
    sys.exit(0)

    # always program defensively, so as not to be surprised

def main():

    a = ["a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", " "]

    d = ["f", "p", "s", "t", "o", "m", "z", "a", "d", "v", "x", "e", "l", "r", "w", "g", " ", "h", "j", "b", "i", "n", "u", "c", "y", "k", "q"]

    # Now when Larry enters a message he wants to encode, all his characters must come from list a[]
    # So his encoder simply looks up the dictionary d[] and replaces character a[j] by d[j]
    # then prints the strange sequence of characters and sends it off to Moe.

    msg = input("Please enter Larry's message: ")

    coded_msg = []

    for j in range(0, len(msg), 1):
        index = search(a, msg[j])
        coded_msg.append(d[index])

    cm = " ".join(coded_msg)

    print(" ")
    print("Moe gets encoded message: ", cm)