V. Rego,

Jan 19, 2016

Week 2, Examples 1

---

#0.py

# a simple function that pauses my code; we'll use it below

def wait():
    c = input() # really, we'll just hit return to continue past "input", and not really care about c

---

#1.py

# A computer word is 32 bits or 64 bits (depending on processor)
# Each bit can hold a 0 or a 1
# A byte is 8 bits

# Every decimal integer can be converted into a binary integer.
# E.g., 12 (an integer) in decimal is represented as 1100 in binary

# Similarly a number like 12.3 (which is called a floating point number)
# will have a binary representation in memory, just a bit more elaborate
# than that mentioned above.

# So now we have TWO data types (integers and floating point numbers, or floats)

# RULE: the data type of an object decides
# -- what values it can contain, and
# -- what operations can be done on it.

# Let's look at these data types and do some operations

def main():
    wait()
    print("type(7) is ", type(7)) # what data type is number 7?
    wait()
    print("type(7.0) is ", type(7.0)) # what data type is number 7.0?
```python
wait()
print("type(7.4) is ", type(7.4))  #what data type is number 7.4?

somenumber = 11

wait()
print("somenumber is ", type(somenumber))  #what data type is somenumber?

somenumber = 11.1  #changing the data type

wait()
print("somenumber is ", type(somenumber))  # and now?

#NOTE: integers can be represented EXACTLY
# but floating point numbers are sometimes only approximately represented

two_thirds = 2/3  # 2 times .3333333333333333... = .6666666666666666....

# What does Python give you?

wait()
print ("two thirds = ", two_thirds, "   and notice it was truncated")

# Let's try another number

x = 10/3

wait()
# the representation is not exact, only approx
print (" 10/3 = ", x,"     why? Because all floats are approximations")

#Note: When operator is not "/", what is result data type?
    # apply operator to int get int result, apply operator to float get float result
    # When operator is "/", you get always float as result

# Now for two more operators
# First "/". a//b = how many times b goes into a (answer is a whole number)```
# Second "%". \( a \% b \) = after \( b \) goes into \( a \) some number of times, what is remainder?

\[
a = 11 \\
b = 3
\]

```python
wait()
print(" a//b = ", a,"//",b, " = ",a//b)
wait()
print(" a%b = ", a,"%",b, " = ", a%b)
wait()
print(" so clearly, a = (a//b)*b + (a%b)")
```

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

#2.py

# a simple function that pauses my code; we'll use it below

def wait():
    c = input()  # really, we'll just hit return to continue past "input"

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

import math  #this lets main access the math library

#Let's try some math formulas (see table 3.2 in textbook)

def main():
    #ceiling: ceiling (7.4) = 8

    wait()
    y = 7.4
    x = math.ceil(y)
    print("ceil(" ,y, ") = ",x)

    #floor: floor(7.4) = 7

    wait()
    x = math.floor(y)  # Note: y is still 7.4
    print("floor(" ,y, ") = ",x)

    wait()
    print("abs(-7.4) = ",math.fabs(-7.4))
wait()  # try 2 raised to power 10
print("2**10 = ",math.pow(2,10))

wait()  # how about factorial 10?
print("factorial(10) = ",math.factorial(10))

# and square root?

wait()
x = 256
print("sqrt(" ,x," ) = ",math.sqrt(256))

#---------------------------------------------------------------------------
#3.py
#Let's solve a quadratic equation
# Equation: a*(x*x) + b*x + c = 0
# Roots are: [ -b +/- sqrt(b*b - 4*a*c)] / 2*a
import math
def main():
    a,b,c = eval(input("Enter a,b,c: "))
    print ("Equation: ",a,"*x**2 + ",b,"*x + ",c," = 0")
    d = b*b - 4*a*c
    if (d < 0):
        print ("discriminant is negative")
    sqd = math.sqrt(d)  # calling the library function just once
    r1 = (-b + sqd)/(2*a)  # but using the result twice here
    r2 = (-b - sqd)/(2*a)
    print(" Solution: r1 = ",r1,"      r2 = ",r2)

# Note: When d < 0 the sqrt function will fail
# now we are dealing with complex numbers and have to use another library

#---------------------------------------------------------------------------
#4.py
#A function to compute factorials

# n! = n*(n-1)*(n-1)*.......*3*2*1

#so 8! = 8*7*6*5*4*3*2*1   (really, the *1 at end is not important)

```python
def fact_b(n):
    # we used "_b" at the end of "fact" because we are 
    # starting the computation at the bottom, i.e., at 1

    ans = 1  # let's start at bottom and multiply
    # because we "accumulate" the answer in this way thru 
    # repeated multiplication in variable ans, 
    # ans is called an "accumulator"

    for i in range(n):
        ans = ans * (i+1)  # i goes from 0 to (-1), so add 1 to it
        print(" ans = ",ans)

    return(ans)  #the function fact returns result ans to whoever calls it
```

def main():

    m = eval(input("Enter value for factorial: "))

    result = fact_b(m)

    print("\n Answer is: ",result)  #\n gives a blank before printing

```

# How about a factorial function that starts at the top?

```python
def fact_t(n):
    ans = n  # let's start at top and multiply
    # because we "accumulate" the answer in this way thru 
    # repeated multiplication, its called an "accumulator"

    for i in range(n-1,1,-1): #start at (n-1), go downwards, stop at 2
        ans = ans * i
```

```python
print(" ans = ", ans)
# the print is part of the loop and will help us see it work

return(ans)  # the function fact returns result ans to whoever calls it

# See page 64,65 of textbook for why "range" must be a sequence
# and one type of sequence is a "list"

#5.py
# The Fibonacci sequence
# 1,1,2,3,5,8,13,21,34,55,......
# F(1) = 1, F(2) = 1
# F(n) = F(n-1) + F(n-2)      for n = 3,4,5,....

def fib(n):
    # this returns the n-th Fib number
    if n == 1:
        # "a == b" means check if a has the same value as b
        return 1
        # the first Fib number is 1
    if n == 2:
        return 1
        # the second Fib number is also 1
    lower = 1
    # the lower of the pair and the higher of the pair
    higher = 1
    for i in range(3,n+1,1):
        # to get the 3rd Fib number onwards ....
        next = lower + higher
        # now redefine lower and higher by shifting up by one
        lower = higher
        higher = next
    return next

# Now get the first n Fib numbers
```

def main():
    n = eval(input("How many Fibonacci numbers do you want? "))

    print("\n\n")

    for i in range(1,n+1,1):
        print("Fib(" ,i," ) = ",fib(i))

#6.py

# a simple function that pauses my code; we'll use it below

def wait():
    c = input()   # really, we'll just hit return to continue past "input"

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

# Simple examples of data "type conversion" and "rounding"

# x = float <> float       means x is a float, where <> is some operator
# x = int <> int            means x is int, unless <> is /
# But what happens with
# z = 7.0 * 3               i.e., float <> int
# either (a) make 7.0 an int, i.e., 7, and thus z is an int, or
#        (b) make 3 a float, i.e., 3.0, and thus z is a float
# floats can represent much larger ranges (bigger/smaller numbers) than
# ints
# so if you did (a), you can lose information (a problem!)
# So Python does (b). It does the type conversion itself

import math

def main():
    a = 7.0 * 3
    wait()

    print ("data type of a = ",a," is ",type(a))

#NOTE: you can do *explicit* type-casting yourself; it is useful at times
x = int(7.7)  # floor or truncation to int
wait()
print ("data type of x = ",x," is ",type(x))

y = float(6)  # int becomes a float
wait()
print ("data type of y = ",y," is ",type(y))

z = round(5.3)  # this is NOT typecasting, simply rounding
wait()  # to the NEAREST WHOLE NUMBER; less than .5 =>
it uses lower value
print ("data type of z = ",z," is ",type(z))
p = int(float(4))  # you can apply one on another
wait()
print ("data type of z = ",p," is ",type(p))
wait()

print(" pi = ", math.pi, " is exact value of pi to default # decimal places")
wait()
print("\n Now see how print statement does rounding\n")
wait()
print("round(math.pi,2) = ",round(math.pi,2))
wait()
print("round(math.pi,3) = ",round(math.pi,3))
wait()

# You'll see that "print" was smart, and gave you only 2 or 3 decimal # places after rounding

#--------------------------------------------------------------------------
#7.py

# We want n random numbers (integers) from the interval [0,100000]

import random

def rand(n):
    for i in range(n):
        r = random.randrange(1,100000)
        print("random number ",i+1," : ",r)

def nrand(): # this is just rand normalized, result is in (0,1)
    # we'll omit the print statement to avoid clutter
    # and we'll just return one random number instead of n
    r = random.randrange(1,100000)/100000
    return(r)

# Let's estimate the value of pi using random numbers

# see http://www.coe.utah.edu/~hodgson/Monte_Carlo.html

# 1. Draw a circle of radius 1, centered at the origin
# 2. Focus only on first quadrant, draw a square of side 1 containing
# part of circle in quadrant 1
# 3. Area of this part of circle is pi/4
# 4. Area of square is 1
# 5. q = Area of this part of circle / area of square = pi/4

# 5. Throw darts (generate random points) at square
# 6. Estimate r = number of darts falling in circle part/ total # of darts
# 7. Because r = pi/4, we get pi = 4*r

import math

def pi(n): # n is the number of darts we'll throw
    count = 0 # number of darts falling inside circle part

    for i in range(n):
        x = nrand() # x coordinate of dart
        y = nrand() # y coordinate of dart

        # now check if dart falls inside circle part, using equation of
        # circle x**2 + y**2 = 1
        if x**2 + y**2 < 1:
count = count + 1
print("Exact value of pi = ", math.pi, 
)print("The simulation gives ....")r = count/nestimate = 4*rreturn(estimate)