CIS 122 Fall 2015 Project 5–functions
Due Monday, Feb 8, 11:59 PM

Briefly

Submit four Python 3 programs
Your programs are worth a total of 25 points

Test your programs -- did they work right? -- before uploading to Canvas.

Your programs should contain code that tests your functions.

P5-function.py
Explore how to call a function of one "argument"
Local variables - the variables created by your function
Are they available to the main program?

P5-utility.py
Some "utility functions" - functions to simplify common error–prone tasks
Build them, test them, use them in any project from this point on.

Also functions to simplify your turtle graphics.
```
jump(distance) # for example
```

P5-display.py
Add some functions to simplify pleasant display of numbers and text. Without these functions:
```
6 John 6783.2
19 Joan 7624.95
216 Jo 13432.57
```

Using the functions you get instead:
```
  6 John     6,783.20
 19 Joan     7,624.95
216 Jo      13,432.57
```

P5-convert.py
Convert dollars to euros, euros to dollars.
Convert temperatures in degrees Celsius to degrees Fahrenheit, and degrees F to degrees C.

Project5-functions.py 5 points:
Learning objective
Practice making your own functions.

Work with **local variables** (variables created within a function). Are those local variables available to your main program?

What happens if a main program has a variable with the **same name** as a variable in your function? Is that legal? Does it cause problems?

Look at a little function (items in blue are required)

```
def xray(volts):  # Define function xray
    """ Returns volts * dirac_number
    """
    dirac_number = 137.00006
    result = volts * dirac_number
    return result

# Call the function

z = 22
gestalt = xray(z) # Python copies argument
                  # volts = z

bongo = 3
chorus = xray(bongo) # copies volts = bongo
```

Short functions

easy to **understand**
easy to **test**
easy to **re–use**

Short functions

do just one thing

Avoid "Swiss Army Knife" functions that do many things

- Features include a large blade, a small blade, a can opener with small screwdriver, a bottle opener with large screwdriver and wire stripper
- Also features scissors, pliers with wire cutter and wire crimper, a wood saw, fish scaler with hook disgorger and ruler, magnifying glass, corkscrew
- Includes a metal saw with metal file, nail file and nail cleaner, reamer with sewing eye, Phillips screwdriver, fine screwdriver, mini screwdriver
- As if that wasn't enough, the Swiss Army Champ knife also sports a hook, wood chisel, ballpoint pen, straight pin, toothpick, tweezers and key ring

Define first, then call

Define once, call many times

Like a recipe for apple pie; you can use the recipe to bake a pie; then later, use the same recipe to bake another pie.

5 points for P5-function.py

Define an **ask_for_int(hint)** function, test features of functions

It asks for input from user using the hint, converts input string to int and returns the int value.

2 points define and test ask_for_int function

```
def ask_for_int(hint):
    """ Return int value of calling input"""
    temp = input(hint)
    value = int(temp)
    return value
# test ask_for_int
height_hint = "Type your height in inches "
height = ask_for_int(height_hint)
print("Height", height, "inches")
```

Since Python copies each argument such as **height_hint** to the names between ( ) in the function def

```
    hint = height_hint
```

you can use any name you want to when calling ask_for_int

1 point

What happens if you try to print **value** ?

print a description such as
print("Trying to print value gets an error") or

print("Trying to print value shows value from the ask_for_int")

Comment out the call to print(value) if it caused an error.

2 points

When both your main program and your function assign a value to a variable name, does the function change the value of a variable in your main program?

```
temp = 27
print("temp", temp)
hint2 = "Lucky number (1-10): "
lucky = ask_for_int(hint2)
print("temp", temp)
```

Did the call to the function change **temp**?

print("temp was changed / unchanged by call to ask_for_int")

```
def ask_for_int(hint):
    """ Return int value of calling input"""
    temp = input(hint)
    value = int(temp)
    return value
```

Here **hint**, **temp** and **value** are all **local variables** defined only while the function is running.

P5-utility.py 5 points

Some "utility functions" - functions to simplify common error–prone tasks
Build them, test them, use them in any project from this point on.

Also functions to simplify your turtle graphics.
`draw_triangle(size  ) # for example`

0 point (you already have this)
`def ask_for_int(hint):`
Displays hint, returns input from user converted to int (whole number)

1 point
`def ask_for_float(hint):`
Displays hint, returns input from user converted to float (decimal number)

1 point
`def ask_for_str(hint):`
Displays hint, returns string input from user

1 point bonus
Each of the "ask_for" functions makes sure **hint** ends in a **blank**.

1 point
`def make_money(amount):`
Accepts numeric amount and returns amount rounded to 2 decimal places.

1 point
`def jump(distance):`
Move turtle forward **without leaving a mark**.
Turtle should be set to mark after leaving the function.

1 point
`def jump_to(x,y):`
Move turtle to location (x,y) **without leaving a mark**.
Turtle should be set to mark after leaving the function.

Hint: `t.goto(x, y)` will be useful here.

Your main program should call each of these functions at least once, somewhat like this (with user typing shown in blue)

```
Enter number 7
You entered 7, next number is 8

Enter a decimal number with 3 decimals 6.789

Type your first name: Morgan
```

Convert the decimal number to money with your `make_money` *function.*

```
Morgan -- money value of 6.789 is 6.79
```

P5-display.py 6 points

Functions to make nice reports that let data line up in readable columns.

A few items to note.

```
x = 12345
show_x = format(x, '8,d')
#'  12,345' # show_x is a string length 8
# ========

money = 123456.78
show_money = format(money, '12,.2f')
```

To form a string like '12,.2f' given width w and decimals d

```
my_format = str(w) + ",." +str(d) + "f"
show_money = format(money, my_format)
```

2 points
`def nice_int(number, width):`
Returns a string showing number value with commas that fits in an area of width spaces

Example
`show_number = nice_int(3025, 6)`
returns `' 3,025'`

2 points
`def nice_float(number, width, decimals):`
Returns a string show a number value with commas that fits in an area of width spaces a decimal point and requested number of decimals.

Example
```
sales = 13034.76
show_sales = nice_float(sales,12,2)
```
returns `'   13,034.76'`

1 points
To left justify a string **s** in a field of width **w**, your function can do this

```
show_str = s.ljust(w)  # l as in "left"
```

`def nice_left_str(string, width):`
Returns a string left justified in a string of width characters, padded on right with blanks.

Example
```
state = "Oregon"
show_state = nice_left_str(state, 16)
```
returns `'Oregon          '`

```
state = "Mississippi"
show_state = nice_left_str(state, 16)
```
returns `'Mississippi     '`

Taken together, these allow nice looking reports:
Instead of

```
2 Oregon 2475 45678.0
12 Utah 3450 12956.2
48 North Carolina 178 4567.33
```

You get something like this

```
  2 Oregon            2,475    45,678.00
 12 Utah              3,450    12,956.20
 48 North Carolina      178     4,567.33
```

1 point Print a similar short report lining up strings and numbers

P5-convert.py 9 points

For the next two functions assume a conversion rate for euros and dollars

rate = 1.09 dollars per 1.00 euros

If you have e **euros mulitply by 1.09 dollars**

$$\frac{}{\textbf{1.00 euros}}$$

If you have d **dollars mulitply by 1.00 euros**

$$\frac{}{\textbf{1.09 dollars}}$$

Notice how euros cancel out in the first formula; dollars cancel out in the 2nd formula.

2 points

```
def euros_to_dollars(euros):

    # put code here to convert to dollars
    return dollars
```

2 points

```
def dollars_to_euros(dollars):

    # put code here to convert to euros
    return euros
```

1 point

Call your functions to test your formulas.

Roughly, 13 dollars gets a little more than 10 euros.
Roughly, 10 euros gives a little more than 12 dollars.

Temperature facts

Freeze ice	0 C	==	32 F
Boil water	100 C	==	212 F
Difference	100 C		180 F

If you have c degrees C multiply by 180F

$$\frac{}{100C}$$

The C's cancel out leaving F degrees above freezing,
then add the F freezing temperature + 32 F

2 points

```
def fahr_to_celsius(f_temp):

    # put code here to convert to c_temp
    return c_temp
```

If you have f degrees:
Start by getting the F temps to a same starting point as Celsius
Subtract 32 F from the f degrees
Then convert F degrees above freezing to C degrees above freezing
    f degrees F above freezing multiply by 100C

$$\frac{}{180F}$$

The F's cancel out leaving C degrees above freezing,

2 points

```
def fahr_to_celsius(f_temp):

    # put code here to convert to c_temp
    return c_temp
```

Test your functions.

0 C should give 32 F

32F should give 0 C

100 C should give 212 F

212 F should give 100 C