Announcements

• Reminders:
  – Program #2 due on Thursday, Sept. 14th by 11:55pm
  – Keep up with Zybooks assignments

Practice from Last Time

Functions

• A function is a group of statements to which we assign a name.
  – Use the “def” keyword to define a function.
• That group of statements can then be referred to by that name later in the program.
  – Call a function by using its name with open/close parenthesis after it.
Function Example

This program has two functions. First we define the main function.

```python
# Function definitions
def main():
    print('I have a message for you."
    message = "Goodbye!"

# Function call
def message():
    print('I am Arthur')
    print('King of the Britons."

# Call the main function.
main()
```

Output:

```
I have a message for you.
I am Arthur
King of the Britons. Goodbye!
```

Example

This program doesn't work!

```python
# This program doesn't work!
def sing_song():
    print("Happy birthday to you!"
    print("Happy birthday dear", name,
    "Happy birthday to you!

    def main():
        name = input("What is your name? "
    sing_song()
    main()
```

`name` is a local variable—it is invisible to Python outside of the main function.

Arguments and Parameters

<table>
<thead>
<tr>
<th>Defining</th>
<th>Defining</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>def name_of_function(var1, var2, ...):</code></td>
<td><code>def name_of_function(param1, param2, ...):</code></td>
</tr>
<tr>
<td><code>statement</code></td>
<td><code>statement</code></td>
</tr>
<tr>
<td><code>statement</code></td>
<td><code>statement</code></td>
</tr>
</tbody>
</table>

When a function is called, all the values inside the parentheses from the calling line are **immediately copied** into the variables given in the function definition.

The values being copied from the calling function are called **arguments**. The variables being copied into are called **parameters**.
def sing_song(name):
    print("Happy bday to you, happy bday to you!")
    print("Happy bday dear", name, "happy bday to you")

def main():
    my_name = input("What is your name? ")
    sing_song(my_name)
    twin_name = input("What is your twin's name? ")
    sing_song(twin_name)

main()

When Python runs the red line, it copies the value of my_name into sing_song's variable name.

def sing_song(name):
    print("Happy bday to you, happy bday to you!")
    print("Happy bday dear", name, "happy bday to you")

def main():
    my_name = input("What is your name? ")
    sing_song(my_name)
    twin_name = input("What is your twin's name? ")
    sing_song(twin_name)

main()

When Python runs the blue line, it copies the value of twin_name into sing_song's variable name.

def sing_song(name):
    print("Happy bday to you, happy bday to you!")
    print("Happy bday dear", name, "happy bday to you")

def main():
    name = input("What is your name? ")
    sing_song(name)
    name = input("What is your twin's name? ")
    sing_song(name)

main()

You may use the same variable names in both places, if desired.
Each function then has its own copy of the variable.
There is no permanent link between the variables.

def some_function(x):
    print("Inside the function, x is", x)
    x = 17
    print("Inside the function, x is changed to", x)

def main():
    x = 2
    print("Before the function call, x is", x)
    some_function(x)
    print("After the function call, x is", x)

main()

Output:
Before the function call, x is 2
Inside the function, x is 2
Inside the function, x is 17
After the function call, x is 2
Recap

• There is no permanent connection between the x in main and the x in some_function.
• Arguments are passed one way only from main to some_function when main calls some_function.
  – This copies main’s value of x into some_function’s x.
• Any assignments to x inside of some_function do not come back to main.

Local Variables

• **Local variable**: variable that is assigned a value inside a function
  – Belongs to the function in which it was created
  • Only statements inside that function can access it, error will occur if another function tries to access the variable
• **Scope**: the part of a program in which a variable may be accessed
  – For local variable: function in which created

Local Variables

• A **local variable** cannot be accessed by statements inside its function which precede its creation
• Different functions may have local variables with the same name
  – Each function does not see the other function’s local variables, so no confusion

Parameters = Local Variables

• “That sounds like local variables.”
• Just as local variables are invisible outside of the function that owns them, variables used as parameters inside a function definition are local to that function.
• Parameters in a function definition are really local variables that are created and assigned values automatically when the function is called.
You’ve seen arguments already.

- `name = input("What is your name? ")`
- `x = 5`
- `y = 2`
- `print("x is", x, "y is", y)`
- `print("their sum is", x + y)`

Arguments can be variables, literals, or math expressions.

In Class Example

- Using functions, write a program that prompts the user for 3 numbers and outputs the average of those numbers.

Tricky Example

```python
def mystery(x, z, y):
    print(z, y-x)

def main():
x = 9
y = 2
z = 5
mystery(z, y, x)
mystery(y, x, z)
mystery(x + 2, y - x, y)
main()
```

Global Variables

- **Global variable**: created by assignment statement written outside all the functions
  - Can be accessed by any statement in the program file, including from within a function
- **DO NOT USE GLOBAL VARIABLES!**
  - Global variables making debugging difficult
    - Many locations in the code could be causing a wrong variable value
  - Functions that use global variables are usually dependent on those variables
    - Makes function hard to transfer to another program
  - Global variables make a program hard to understand!
Global Constants

- **Global constant**: global name that references a value that **cannot be changed**
  - OK to use global constants in a program
  - To simulate global constant in Python, create global variable and do not re-declare it within functions

Global Constant Example

```python
# The following is used as a global constant to represent
# the contribution rate.
CONTRIBUTION_RATE = 0.05

def main():
    gross_pay = float(input('Enter the gross pay: '))
    bonus = float(input('Enter the amount of bonus: '))
    show_pay_contrib(gross_pay)
    show_bonus_contrib(bonus)

def show_pay_contrib(gross_pay):
    contrib = gross_pay * CONTRIBUTION_RATE
    print('Contribution for gross pay: $', \
          format(contrib, ',.2f'), \
          sep='')

def show_bonus_contrib(bonus):
    contrib = bonus * CONTRIBUTION_RATE
    print('Contribution for gross pay: $', \
          format(contrib, ',.2f'), \
          sep='')

# Call the main function.
main()
```

Practice

1. Modify singHappyBirthday.py
   - You no longer have a twin. Now you have a sibling that is two years older than you, but you share the same birthday.
   - Edit code so that sing_song now will print the lyrics but also print how old the person is.
   - Add a second parameter to sing_song called age.
   - Edit main() to ask for your age, as well as your name and sibling’s name.
   - Edit the two calls to sing_song so appropriate ages are passed as arguments.

2. Write a new Python program that asks the user to input 2 numbers and outputs the sum of those numbers.
   - Use 2 functions
     * main(): - Prompts the user to enter 2 numbers and calls sum()
     * sum(): - Takes in 2 parameters and outputs the sum of those numbers