COMP 141

Functions that take arguments, local variables

1

Announcements

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

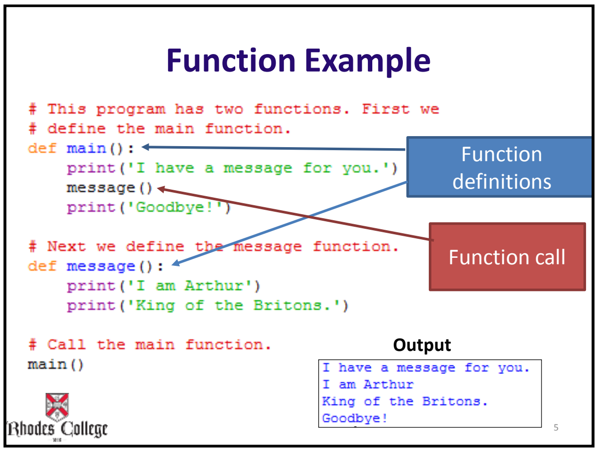
2

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.

4



Function Example

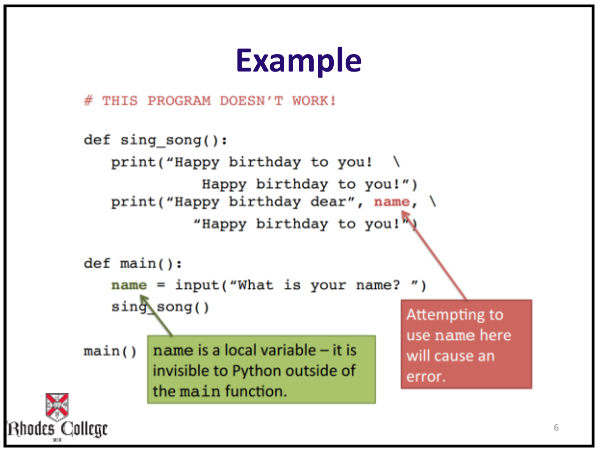
```
# This program has two functions. First we
# define the main function.
def main():
    print('I have a message for you.')
    message()
    print('Goodbye!')

# Next we define the message function.
def message():
    print('I am Arthur')
    print('King of the Britons.')

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

Function definitions
Function call
Rhodes College

Output

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

5



Example

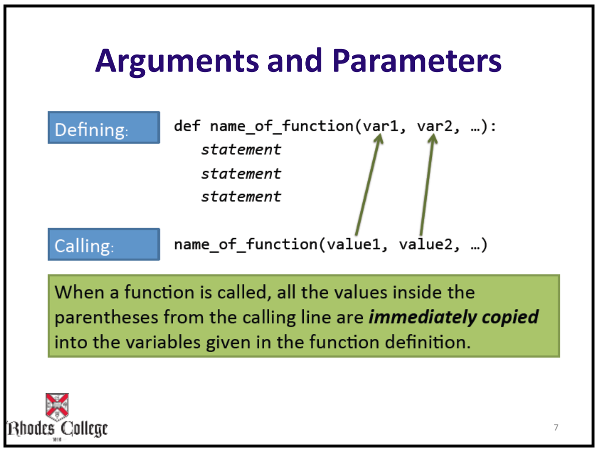
```
# THIS PROGRAM DOESN'T WORK!

def sing_song():
   print("Happy birthday to you!  \
             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.

Attempting to use name here will cause an error.

Rhodes College

6



Arguments and Parameters

Defining:

```
def name_of_function(var1, var2, …):
    statement
    statement
    statement
```

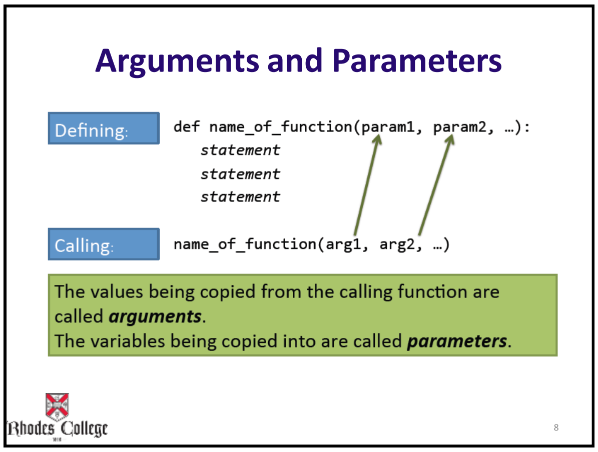
Calling:

```
name_of_function(value1, value2, …)
```

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.

Rhodes College

7



Arguments and Parameters

Defining:

```
def name_of_function(param1, param2, …):
    statement
    statement
    statement
```

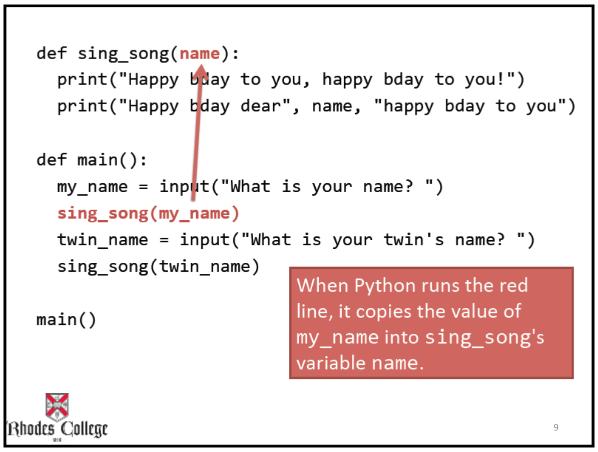
Calling:

```
name_of_function(arg1, arg2, …)
```

The values being copied from the calling function are called ***arguments***.
The variables being copied into are called ***parameters***.

Rhodes College

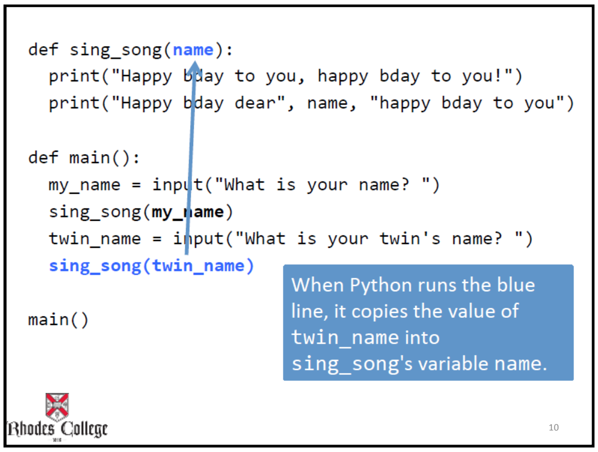
8



```
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.

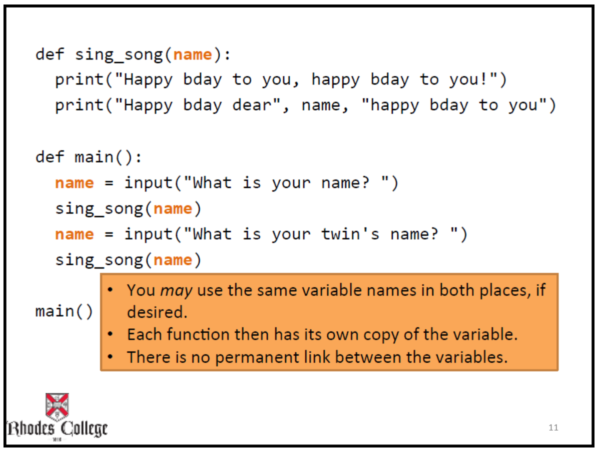
Rhodes College 9



```
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.

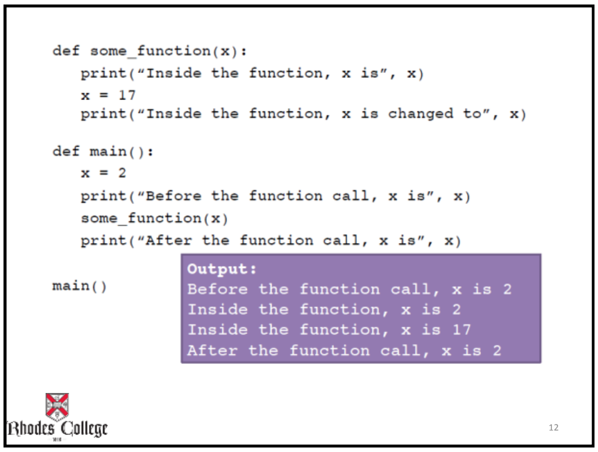
Rhodes College 10



```
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.

Rhodes College 11



```
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
```

Rhodes College 12

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`.

Rhodes College

Local Variables

- **<u>Local variable</u>:** 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
- **<u>Scope</u>:** the part of a program in which a variable may be accessed
  - For local variable: function in which created

Rhodes College

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

Rhodes College

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.

Rhodes College

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.

Rhodes College

In Class Example

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

Rhodes College

Tricky Example

```
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 + z, y - x, y)

main()
```

Rhodes College

Global Variables

- **<u>Global variable</u>:** 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!

Rhodes College

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 bonuses: '))
    show_pay_contrib(gross_pay)
    show_bonus_contrib(bonus)

# The show_pay_contrib function accepts the gross
# pay as an argument and displays the retirement
# contribution for that amount of pay.
def show_pay_contrib(gross):
    contrib = gross * CONTRIBUTION_RATE
    print('Contribution for gross pay: $', \
          format(contrib, ',.2f'), \
          sep='')

# The show_bonus_contrib function accepts the
# bonus amount as an argument and displays the
# retirement contribution for that amount of pay.
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