COMP 204

Functions

Mathieu Blanchette
based on material from Yue Li and Carlos Oliver Gonzalez

Quiz 10 password

Functions: Why we need them

In large programs, we often need to perform several times the same type of computation. Examples:

- Ask the user for some input and check its validity
- Calculate the distance between two points in the plane
- Find the largest element in a list

Until now, the only way we have to do this is to duplicate and adapt code. This is bad because:

- It is very error-prone, hard to debug and maintain
- It makes the program unnecessarily large
- It makes the program hard to read

Example: you use the same distance equation in 10 different programs but later on decide to change the distance calculation.
Functions: Allow re-using a piece of code without duplicating it.
We've used many functions already: print(), sqrt(), isdecimal()..
Today, we learn how to define *our own* functions.

Functions: the first example

```python
# This is the print_welcome function
def print_welcome():
    print("************************")
    print("* Welcome to COMP 204! *")
    print("************************")

# This is now outside the print_welcome function
print_welcome()
print("My name is Mathieu")
# Some more code

#print again
print_welcome()
print("etc ...")
#and again
print_welcome()
```

Notes:

- Use the keyword `def` to define our own functions.
- Once the function is defined, just call it using its name and its code will execute.
- **Note:** without a call, the function's code will not be executed.

The anatomy of a function

```python
# function header
def function_name( function_arguments ):
    # body of function
    # ...
    # ...

# rest of program
```

- Function header
    1. **def** tells Python you are defining a function
    2. **function_name**. Functions are objects so we give them names
    3. **(function_arguments)** Objects you would like the function to work on (optional)
- Function body
    - Any code that is tabbed at least once and follows the **header** is stored in the function.

Functions with arguments

Without arguments, a function always executes the same thing.
For more flexibility, we pass arguments to the function.

```python
# This function welcomes a student to COMP 204
def print_welcome_204(student_name):
    print("Dear",student_name)
    print("Welcome to COMP 204")

# This function welcomes a student to any course
def print_welcome(student_name, course_name):
    print("Dear",student_name)
    print("Welcome to", course_name)


# This is now outside the print_welcome function
print_welcome_204("Yang")
print_welcome_204("Alessandro")
print_welcome("Veronica", "COMP 204 Fall 2019")
```

What happens when a function is called?

When a function is called:

- A new *local* variable is created for each argument (if any)
- The value of each argument variable is initialized to that provided with the function call
- The body of the function is executed. This may include defining/using other local variables.
- When the body is finished executing,
    - We discard local variables
    - We go back to the line where the function was called, and continue execution from there.

Note: A function can call another function. For example: the printWelcome() function calls the print() function.

The `return` statement

Until now, our functions print text, but the result of their computation cannot be communicated to the rest of the program.

- The return statement is a special word that lets the function “emit’ an object.
- This is useful because it lets the code that called the function store the output in a variable and perform operations with it later on.
- **return** is NOT the same as print()
- When Python reaches a return statement it *immediately exits* the function.
- If we reach the end of a function without reaching a return statement, the function returns the empty object None.

Examples of functions

We have used many functions already:

- print(...): prints stuff to screen, returns nothing
- input(...): returns a string from keyboard entry.
- range(...): returns a list of integers
- int(...): returns an integer from a string
- math.sqrt(...): returns the square-root of a number
- and many more...

Example 2: Computing Euclidean distance

```python
import math

# this function calculates the distance between
# two points (x1, y1) and (x2, y2) in Euclidean space
def distance(x1, y1, x2, y2):
    d = math.sqrt( (x1-x2)**2 + (y1-y2)**2 )
    return d
    print("Hello") #this is  never reached

my_distance = distance(3,1,5,7)
print("The distance is", my_distance)

print("The distance is ", distance(3,1, 5,7) )

print(d) # error: d is not accessible
         # outside the distance function
```

Demo in Spyder

- Execute the distance2D.py program in debug mode.
- Learn how to ”Step into function”
- See the local variables.

Functions: Why we need them

Functions are useful because they enable :

- **Code re-use:**
  - Once you've written a function *and made sure it works*, you can re-use it as many times as needed, from any program you want.
  - You can also re-use code written by others
  - Other can re-use you code
- **Encapsulation:**
  - As the user of a function, all you need to know is its name, arguments, and what it outputs. No need to worry about it works.
  - Allows breaking down complex tasks into small, easy to understand subtasks
  - Allows thinking about a problem at a high-level, focussing on the aspects that matter to your project.

```python
import math
def euclid(x_h, y_h, x_a, y_a):
    return math.sqrt((x_h - x_a)**2 + (y_h - y_a)**2)

def evaluate_risk(distance):
    if distance <= 20:
        return "You must evacuate"
    elif distance <= 40:
        pregnant = input("Are you pregnant? (yes/no) ")
        if (pregnant in ["yes","Yes","Y","y"]):
            return "You must evacuate"
        else:
            return "Evacuation is recommended"
    else:
        return "No need to evacuate"

def evacuate_assessment():
    x_acc = float(input("Enter x coord. of nuclear: "))
    y_acc = float(input("Enter y coord. of nuclear: "))
    x_home = float(input("Enter x coordinate of home: "))
    y_home = float(input("Enter y coordinate of home: "))
    distance=euclid(x_home, y_home, x_acc, y_acc)
    message = evaluate_risk(distance)
    print(message)


# our main program starts here
evacuate_assessment()
```

Example 3: Safe input for integers

Goal: Write a function that repeatedly asks a user to enter an integer, until the number entered in within a desired range. Once a valid input has been entered, return that value.

```python
# Asks user to enter a value by printing message
# Repeats until value is between min_val and max_val
def input_in_range(message, min_val, max_val):

    while True: # loops until return statement is executed
        n = int(input(message))
        if n >= min_val and n <= max_val:
            return n
        else:
            print("Number outside range",min_val, max_val)

# our main program starts here
age = input_in_range("Enter age: ",0,150)
height = input_in_range("Enter height (in cm): ",0,250)
```

Example 4: Safe input for strings

Goal: Write a function that repeatedly asks a user to enter a string, until the number entered in within a desired lis of acceptable values. Once a valid input has been entered, return that value.

```python
# Asks user to enter a string value by printing message
# Repeats until value is within list acceptable values
def input_in_list(message, acceptable_list):

    while True: # loops until return statement is executed
        s = input(message)
        if s in acceptable_list: # tests if s is in list
            return s
        else:
            print("Please respond by ",acceptable_list)


history = input_in_list("History of diabetes? ", ["yes","no"])
gender = input_in_list("Gender? ", ["female","male"])
```

Example 5: Checking prime number

- A function body can have multiple return statements. The first one encountered during execution will end the function
- Exercise: write a function that returns True if it is given a prime number and False otherwise.

```python
# This function return True if the integer
# provided as argument is a prime number
def is_prime( n ):
    # look at all candidate factors of n
    for f in range(2, n):
        # see if f is a factor of n
        # by computing the remainder of the division
        if n % f == 0:
            return False

    # if we reach this, it is because we found
    # no factor for n, so it is prime
    return True

if is_prime(int(input("Enter a number: "))):
    print("The number is prime")
else:
    print("The number is not prime")
```

Example (advanced): Recursion: function that calls itself

```python
# a function that calls itself
def count_down_recursion(count):
    if count > 0:
        print(count)
        count_down_recursion(count-1)

count_down_recursion(10)
```