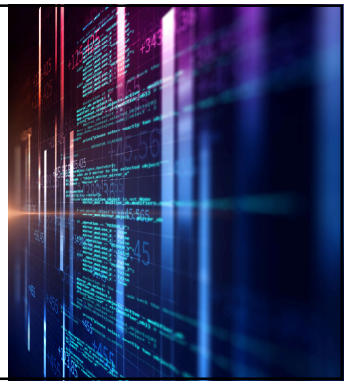
Functions & Modular Programming



Brandon Krakowsky

Penn Engineering

1

Functions

2

What is a Function?

- A *function* is a block of organized, reusable code that is used to perform a single, related action
- A *function* provides better modularity for your applications and a high degree of code reusing
- Python provides *built-in functions*
  - These are part of the core language
- Python also allows you to define your own *user-defined functions*

3

Built-In Functions

- You've already been using built-in functions!
  - The *print* function to print a string
    `print("Hello World!")`
  - The *input* function to get user input
    `input("What is your favorite movie?")`
  - The *int* function to cast from one data type to an integer
    `int(3.1)`
- There are lots of built-in functions. Here are some others:
  - float(*x*) - casts string or integer *x* to a float
  - round(*float*, *int*) - rounds *float* to *int* decimal places
  - max(*arg1*, *arg2*, *argN*) - gets the maximum value of arguments
  - min(*arg1*, *arg2*, *argN*) - gets the minimum value of arguments
  - len(*s*) – gets the length (number of items) of an object *s*

For reference: https://docs.python.org/3/library/functions.html

4

User-Defined Functions

- Functions have conventions
  - Name a function based on what it does
  - Whitespace is important!
    - Function body "code blocks" (groups of statements) have to be indented (4 spaces or tab)
- Sometimes a function takes an input
  - These are called *parameters*
  - When you call (or use) the function, you pass *arguments* to satisfy the *parameters*
- Sometimes a function produces an output
  - This is called the function's *return* value

5

User-Defined Functions

- You define a *function* using the *def* keyword, followed by the *function* name and parenthesis

```
def function_name(param1, …, paramN):
    statements
    return
```

- Parenthesis include optional *parameters*, treating them as variables
- Functions optionally *return* a value

6

User-Defined Functions

- Let's define a function *square*
  - It takes one number as a *parameter*
  - It *returns* the result of squaring that number

```
def square(x):
    y = x * x
    return y
```

- Now let's use the function *square*
  - When we call it, we pass 10 as an *argument*
  - Then we store the *return* value in a result variable and print it

```
to_square = 10
result = square(to_square)
print(result)
```


7

User-Defined Functions

- Let's define a function *greater_than*
  - It takes two numbers as *parameters*
  - It *returns* True if the 1st *parameter* is greater than the 2nd *parameter*

```
def greater_than(x, y):
    if x > y:
        return True
    else:
        return False
```

- Now let's use the function *greater_than*
  - When we call it, we pass 2 and 3 as *arguments*
  - Then we store the *return* value in a result variable and print it

```
a = 2
b = 3
result = greater_than(a, b)
print("{} is greater than {}: {}".format(a, b, result))
```


8

User-Defined Functions - Docstring

- You can (and should) provide a *documentation string* (or *docstring*) for your function
  - A *docstring* describes the operation of the function (or class)
- A *docstring* is for someone who is using your function and wants to know "what it does", at a high level
- This is different from a *comment*, which is for a programmer who might be reading your code and wants to know the details of "how it works"

9

User-Defined Functions - Docstring

- To create a *docstring*, include a string as the first statement in the function definition

```
def greater_than(x, y):
    """Returns True if x is greater than y, otherwise False.
    """
    if x > y:
        return True
    else:
        return False
```


10

User-Defined Functions – Accessing Docstring

- The *docstring* is accessible to a user of your program by getting *help* on the function `help(greater_than)`
- It's also accessible directly `print(greater_than.__doc__)`
  - Note: __doc__ has 2 underscores before and after "doc"

11

User-Defined Functions - Exercise

- Define a function *get_factors* that takes an integer as a *parameter* and *returns* a list of factors of that number
  - Basically, find the numbers between 1 and the given integer that divide the number evenly

```
def get_factors(x):
    """Returns a list of factors of given number.
    """
    factors = []
    #To find the possible factors, check for division by the numbers 1 to x
    for i in range(1, x + 1):
        if (x % i == 0):
            factors.append(i) #append to new list

    return factors

print(get_factors(21))
```


12

User-Defined Functions - Exercise

- Define a function *get_factors* that takes an integer as a *parameter* and *returns* a list of factors of that number
  - Basically, find the numbers between 1 and the given integer that divide the number evenly
- Here's another way to do it, in one line, with list comprehension!

```
def get_factors(x):
    """Returns a list of factors of given number.
    """
    return [i for i in range(1, x + 1) if x % i == 0]

print(get_factors(21))
```


13

User-Defined Functions - Exercise

- Define a function *unique_list* that takes a list of numbers as a *parameter* and *returns* a new list with the unique values
- Call the function with the list [1,2,3,3,3,3,4,5]

```
def unique_list(l):
  """Returns a list of unique values from given list.
  """
  x = []
  for a in l:
      if a not in x:
          x.append(a)

  return x

print(unique_list([1,2,3,3,3,3,4,5]))
```


14

Execution Order

15

Execution Order

- When you load and run a Python *module* (file), the statements and definitions in the file are executed in the order in which they occur
- Executing a *def* defines a function, it doesn't run the function
  - Functions are only run when they are called
- A very small program might not define any functions at all, but just be a series of statements to be executed
- Most programs consist of a lot of function definitions, along with maybe a few *top-level* statements (statements not in functions)
- Usually one particular function is the starting point of a program
  - By convention, it is called *main* (this is mandatory in Java!)
  - For example:

```
def main():
   print('Hello world!')
```


16

Execution Order

- Before executing code in a *module* (file), Python will define a few special variables
  - If the Python interpreter is running the file as the main program, it sets the special __name__ variable to have a value "__main__"
  - If the file is being imported from another module, __name__ will be set to that module's (file's) name
- To direct the Python interpreter when it first reads a file, add the following conditional (to the bottom) of your script:

```
if __name__ == "__main__ ":
  main()
```

- This will run the *main* function, if the file is loaded as the main program
- Note:
  __name__ has 2 underscores before and after "name"
  __main__ has 2 underscores before and after "main"

17

Example Programs

18

Vowel/Word Counter Program

- Create a new script file. Write a program that does the following:
  - Counts the number of vowels in a string.
  - Counts the number of words in a sentence.
- Start by creating a *vowel_counter* function.

```
def vowel_counter(string):
    """Counts the number of vowels in a string.
    """
    vowel_count = 0

    #for each char in string, check if it's in the string of vowels
    for char in string:
        if char in 'aeiou':
            vowel_count += 1

    return vowel_count
```


19

Vowel/Word Counter Program

- Define the *main* function that calls and runs the *vowel_counter* function within a loop. Then run your program.

```
def main():
    """Gets user input of string and prints the count of vowels.
    """
    while 1 == 1: #create an infinite loop

        input_string = input("please give me a string\n")
        #uses \n (new line character) to force input to the next line

        #exit infinite loop by entering '-1'
        if input_string == '-1':
            break

        print(vowel_counter(input_string), "vowels in", input_string)

#To automatically run the main function in your program
if __name__ == '__main__':
    main()
```


20

Vowel/Word Counter Program

- Now create a *word_counter* function.

```
def word_counter(sentence):
    """Counts the number of words in a sentence.
    """
    sentence = sentence.strip() #strips whitespace from beginning and end of entire string
    word_count = 0

    #for each char in sentence, check if it's a space
    space_count = 0
    for char in sentence:
        if char in ' ':
            space_count += 1

    word_count = space_count + 1

    return word_count
```


21

Vowel/Word Counter Program

- Now add a *word_counter* call to the *main* function. Then run your program.

```
def main():
    """Gets user input of string and prints the count of vowels and/or words.
    """
    while 1 == 1: #create an infinite loop

        input_string = input("please give me a string\n")
        #uses \n (new line character) to force input to the next line

        #exit infinite loop by entering '-1'
        if input_string == '-1':
            break

        #print(vowel_counter(input_string), "vowels in", input_string)
        print(word_counter(input_string), "words in", input_string)
```


22

Vowel/Word Counter Program - Refactoring

- Note, both the *vowel_counter* function and the *word_counter* function have a *similar code block*. Let's create a separate function that does just that.
  - This is known as *refactoring* the code

```
def vowel_counter(string):
    """Counts the number of
       vowels in a string.
    """
    vowel_count = 0
    for char in string:
        if char in 'aeiou':
            vowel_count += 1

    return vowel_count
```

```
def word_counter(sentence):
    """Counts the number of
       words in a sentence.
    """
    sentence = sentence.strip()
    word_count = 0

    space_count = 0
    for char in sentence:
        if char in ' ':
            space_count += 1

    word_count = space_count + 1

    return word_count
```


23

Vowel/Word Counter Program - Refactoring

- Create a *count_instance_of_str* function.

```
def count_instance_of_str(string1, string2):
    """Counts characters in string1 that are also in string2.
    """
    count = 0

    #for each char in string1, check if it's in string2
    for char in string1:
        if char in string2:
            count += 1

    return count
```


24

Vowel/Word Counter Program - Refactoring

- Now we can use the *count_instance_of_str* function in our other functions.

```
def word_counter_v2(sentence):
    """Returns the number of words in vowels and/or words..
    """
    sentence = sentence.strip() #strips whitespace from beginning and end of entire string

    #counts the characters in sentence, that are also in ' ' (space)
    num_spaces = count_instance_of_str(sentence, ' ')

    num_words = num_spaces + 1

    return num_words
```


25

Vowel/Word Counter Program - Refactoring

- Now we can use the *count_instance_of_str* function in our other functions.

```
def vowel_counter_v2(string):
    """Counts the number of vowels in a string.
    """
    #counts the characters in string, that are also in 'aeiou'
    num_vowels = count_instance_of_str(string, 'aeiou')

    return num_vowels
```


26

Vowel/Word Counter Program

- Now add the new calls to the *main* function. Then run your program.

```
def main():
    """Gets user input of string and prints the count of vowels and/or words.
    """
    while 1 == 1: #create an infinite loop

        input_string = input("please give me a string\n")
        #uses \n (new line character) to force input to the next line

        #exit infinite loop by entering '-1'
        if input_string == '-1':
            break

        #print(vowel_counter(input_string), "vowels in", input_string)
        #print(word_counter(input_string), "words in", input_string)

        print(vowel_counter_v2(input_string), " vowels in ", input_string)
        print(word_counter_v2(input_string), " words in ", input_string)
```


27