Designing a Program to Solve Your Problem (2/2): Functions
Identify Key Points in Program

Input: freeform text

Computer’s choice: rock, paper or scissors

User’s choice: rock, paper, scissors or invalid?

Winner: user or computer?

“Invalid input”
Identify Functionality Blocks and Connect Inputs and Outputs

Input: freeform text

Computer’s choice?

User’s choice?

Winner: user or computer?

Determine Computer’s Choice
Input: None
Output: Computer’s Choice

Determine User’s Choice
Input: Freeform text
Output: User’s Choice

Decide Winner
Input: User’s & Computer’s Choice
Output: Winner

Format Output
Input: Winner

Output: “Win/Lose/Draw”
“Invalid Input”
Identify Functionality Blocks and Connect Inputs and Outputs

Input: freeform text

Computer’s choice?

User’s choice?

Winner: user or computer?

Determine Computer’s Choice
Input: None
Output: Computer’s Choice

Determine User’s Choice
Input: Freeform text
Output: User’s Choice

Decide Winner
Input: User’s & Computer’s Choice
Output: Winner

Format Output
Input: Winner

Output: “Win/Lose/Draw” “Invalid Input”
Defining Your Own Functions

- Functions are like small programs inside your full program
- They have inputs and compute outputs
- Indented code after function definition is part of function
- Remember the colons!
Function Skeleton Code

Every function definition starts with `def`

```python
def function_name(param1, param2):
    # The logic of the function will go here
    return output_variable
```

The function name

Input Variables

The output of the function is placed after the word `return`
def function_name(param1, param2):
    # The logic of the function will go here
    return output_variable

The input variables are called parameters

The actual values of parameters are called arguments

Don’t forget the colon!

The output value returned is called the return value
Designing Functions
• When you write functions, they should conform to a **contract**

• Your code that calls these functions expects them to conform to the contract

• By creating a clear purpose for this function, it’s use inside your program remains clear

---

**Contract**

*Every contract has three parts...*

\[ \text{function name} : \text{domain} \rightarrow \text{range} \]

Domain=types of possible input (string, int, Boolean, etc.). Range=types of potential output.
Purpose Statement

• Describe in words what the function does

What does the function do?
Examples

Provide some example inputs and outputs that fit your contract and make sense with what you want your function to do...

EXAMPLE: def \( (\quad \quad \quad \quad) \): \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad output

EXAMPLE: def \( \quad \quad \quad (\quad \quad \quad \quad) \): \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad output

EXAMPLE: def \( \quad \quad \quad (\quad \quad \quad \quad) \): \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad output
Definition

• After you have a clear idea of what the function is supposed to do and how it will work

• Write your definition, as it would look in Python code, with a return statement of the output variable

```
def ___________(___________):
    function name variable name(s)
    return ____________________________

    variable name for something in the range

This is your gateway to writing actual code
```
Function Documentation

- Functions require their own special documentation called **docstrings**

- Docstrings communicate the information you wrote in your worksheet
Docstrings

def function_name(param1, param2,...):
    """ The purpose statement of the function on the first line

    Keyword arguments:
    param1 -- Type and expected values of input_parameter1
    param2 -- Type and expected values of input_parameter2

    Returns:
    output_variable The type and expected values of output_variable
    """

    # Function code goes here
    return output_variable
Function Skeleton Code

def function_name(param1, param2):
    """ The purpose statement of the function on the first line

The purpose statement goes at the top"""
Function Skeleton Code

def function_name(param1, param2):
    """ The purpose statement of the function on the first line

    Keyword arguments:
    param1 -- Type and expected values of input_parameter1
    param2 -- Type and expected values of input_parameter2

    Function arguments are listed on separate lines
Function Skeleton Code

def function_name(param1, param2):
    
    """ The purpose statement of the function on the first line
    
    Keyword arguments:
    param1 -- Type and expected values of input_parameter1
    param2 -- Type and expected values of input_parameter2
    
    Returns:
    output_variable The type and expected values of output_variable

    Then return result is the last line
Function Skeleton Code

def function_name(param1, param2):
    """ The purpose statement of the function on the first line

    Keyword arguments:
    param1 -- Type and expected values of input_parameter1
    param2 -- Type and expected values of input_parameter2

    Returns:
    output_variable The type and expected values of output_variable
    """

Don’t forget the closing quotes!
def function_name(param1, param2):
    """ The purpose statement of the function on the first line
    
    Keyword arguments:
    param1 -- Type and expected values of input_parameter1
    param2 -- Type and expected values of input_parameter2
    
    Returns:
    output_variable The type and expected values of output_variable
    """

    # Function code goes here
    return output_variable
The main function

- If all your code is now in functions, where does your program start?
- We use a special function called `main` as our starting point
- Place this code at the very bottom of your code

```python
def main():
    # Put highest level logic here

if __name__ == '__main__':
    main()
```
def plus(variable_1, variable_2):
    result = variable_1 + variable_2
    return result

def main():
    a = 3
    b = 5
    c = plus(a, b)

if __name__ == "__main__":
    main()
def plus(variable_1, variable_2):
    result = variable_1 + variable_2
    return result

def main():
    a = 3
    b = 5
    c = plus(a, b)

if __name__ == "__main__":
    main()
def plus(variable_1, variable_2):
    result = variable_1 + variable_2
    return result

def main():
    a = 3
    b = 5
    c = plus(a, b)

if __name__ == "__main__":
    main()
def plus(variable_1, variable_2):
    result = variable_1 + variable_2
    return result

def main():
    a = 3
    b = 5
    c = plus(a, b)

if __name__ == "__main__":
    main()
def plus(variable_1, variable_2):
    result = variable_1 + variable_2
    return result

def main():
    a = 3
    b = 5
    c = plus(a, b)

    if __name__ == "__main__":
        main()
Python’s Memory Table: Function Edition

```python
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(c)

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>main()</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(c)

if __name__ == "__main__":
    main()
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(c)

if __name__ == "__main__":
    main()
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(c)

if __name__ == "__main__":
    main()
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(c)

if __name__ == "__main__":
    main()
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(c)

if __name__ == "__main__":
    main()
```python
def plus(a, b):
    a = 2 * a
    b = 3 + b
    c = a + b
    return c

def main():
    a = 3
    b = 5
    c = plus(a, b)
    c += 1
    print(c)

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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>int</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>2</td>
<td>int</td>
<td>5</td>
</tr>
<tr>
<td>c</td>
<td>5</td>
<td>int</td>
<td>14</td>
</tr>
</tbody>
</table>
def plus(a, b):
a = 2 * a
b = 3 + b
c = a + b
return c

def main():
a = 3
b = 5
c = plus(a, b)
c += 1
print(c)

if __name__ == "__main__":
    main()
Writing to global variables in functions will lead to problems

count = 0

def some_function():
    count += 1
    # Do some function stuff

def main():
    some_function()
    some_function()
    some_function()
    some_function()
    print(count)

if __name__ == "__main__":
    main()
Writing to global variables in functions will lead to problems

count = 0

def some_function():
    count = count + 1
    # Do some function stuff

def main():
    some_function()
    some_function()
    some_function()
    some_function()
    print(count)

if __name__ == "__main__":
    main()
Writing to global variables in functions will lead to problems

count = 0
def some_function():
    count = count + 1
    # Do some function stuff
def main():
    some_function()
    some_function()
    some_function()
    some_function()
    print(count)

if __name__ == "__main__":
    main()
Writing to global variables in functions will lead to problems

count = 0
def some_function():
    count = count + 1
    # Do some function stuff
def main():
    some_function()
    some_function()
    some_function()
    some_function()
    print(count)
if __name__ == "__main__":
    main()
Writing to global variables in functions will lead to problems

count = 0
def some_function():
    count = count + 1
    # Do some function stuff
def main():
    some_function()
    some_function()
    some_function()
    some_function()
    print(count)

if __name__ == "__main__":
    main()
Writing to global variables in functions will lead to problems

count = 0
def some_function():
    count = count + 1
    # Do some function stuff
def main():
    some_function()
    some_function()
    some_function()
    some_function()
    print(count)

if __name__ == "__main__":
    main()
Writing to global variables in functions will lead to problems

count = 0

def some_function():
    count = count + 1
    # Do some function stuff

def main():
    some_function()
    some_function()
    some_function()
    print(count)

if __name__ == "__main__":
    main()
Writing to global variables in functions will lead to problems

```python
count = 0
def some_function():
    count = count + 1
    # Do some function stuff
def main():
    some_function()
    some_function()
    some_function()
    print(count)
if __name__ == "__main__":
    main()
```
Writing to global variables in functions will lead to problems

count = 0
def some_function():
    count = count + 1
    # Do some function stuff
def main():
    some_function()
    some_function()
    some_function()
    print(count)

if __name__ == "__main__":
    main()
Writing to global variables in functions will lead to problems

count = 0
def some_function():
    count = count + 1
    # Do some function stuff
def main():
    some_function()
    some_function()
    some_function()
    print(count)

if __name__ == '"__main__":
    main()
Writing to global variables in functions will lead to problems

count = 0

def some_function():
    count = count + 1
    # Do some function stuff

def main():
    some_function()
    some_function()
    some_function()
    print(count)

if __name__ == "__main__":
    main()
Writing to global variables in functions will lead to problems

count = 0

```python
def some_function():
    count = count + 1
    # Do some function stuff

def main():
    some_function()
    some_function()
    some_function()
    some_function()
    print(count)

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

globals

<table>
<thead>
<tr>
<th>Variable</th>
<th>Address</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>1</td>
<td>int</td>
<td>0</td>
</tr>
</tbody>
</table>
Writing to global variables in functions will lead to problems

count = 0

def some_function():
    count = count + 1
    # Do some function stuff

def main():
    some_function()
    some_function()
    some_function()
    print(count)

if __name__ == "__main__":
    main()
Writing to global variables in functions will lead to problems

count = 0

def some_function():
    count = count + 1
    # Do some function stuff

def main():
    some_function()
    some_function()
    some_function()
    some_function()
    print(count)

if __name__ == "__main__":
    main()
Writing to global variables in functions will lead to problems

count = 0

def some_function():
    count = count + 1
    # Do some function stuff

def main():
    count += 1
    some_function()
    some_function()
    some_function()
    print(count)

if __name__ == "__main__":
    main()
Writing to global variables in functions will lead to problems

count = 0

def some_function():
    count = count + 1
    # Do some function stuff

def main():
    count += 1
    some_function()
    some_function()
    some_function()
    print(count)

if __name__ == "__main__":
    main()
Variable Scope

• Variables defined within a function only exist temporarily

• Variables defined outside of a function are called globals

• Unless handled specially, global variables are read only (keep them that way!)

• To avoid confusion between local variables and global variables, add underscores before and after a global variable name (_pounds_per_kilogram_, _pandas_per_lemur_)
Global Variables in Programs

```python
#!/usr/bin/env python3
_pounds_per_kilogram_ = 2.2

def main():
    weight_in_pounds = input("What is your weight? ")
    weight_in_kilograms = weight_in_pounds / _pounds_per_kilogram_
    print("You weigh {} kilograms".format(weight_in_kilograms))

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

• Use functions to modularize your code

• This allows you to reason and think about smaller blocks of code at the same time

• Document your functions to communicate what they are doing

• Don’t forget the colons!