Functions: Decomposition And Code Reuse, Part 1

- Defining new functions
- Calling functions you have defined
- Declaring variables that are local to a function

Tip For Success: Reminder

- Look through the examples and notes before class.
- This is especially important for this section because the execution of these programs will not be sequential order.
- Instead execution will appear to 'jump around' so it will be harder to follow the examples if you don't do a little preparatory work.

- Also it would be helpful to take notes that include greater detail:
  - For example: Literally just sketching out the diagrams that I draw without the extra accompanying verbal description that I provide in class probably won't be useful to study from later.

James Tam

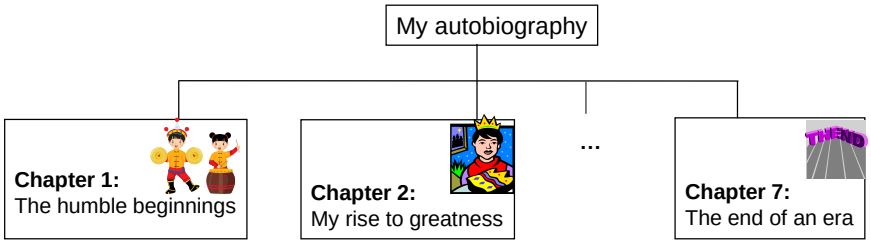
Solving Larger Problems

- Sometimes you will have to write a program for a large and/or complex problem.
- One technique employed in this type of situation is the top down approach to design.
  - The main advantage is that it reduces the complexity of the problem because you only have to work on it a portion at a time.

James Tam

Top Down Design

1. Start by outlining the major parts (structure)



2. Then implement the solution for each part

Chapter 1: The humble beginnings

It all started ten and one score years ago with a log-shaped computer work station…

Image copyright unknown

Applying The Top Down Design To Programming

- First: outline the parts of your program before writing the instructions.
  - These 'parts' will take the form of functions.
- Second: implement (write) the code for one part/function at a time.
- Third: run a reasonable number of tests on that function to ensure it is correct.
- Fourth: apply any bug fixes that may be needed and test again.
- Fifth: only after a reasonable amount of testing has been done on a function should Steps 2 – 4 be applied on another function.

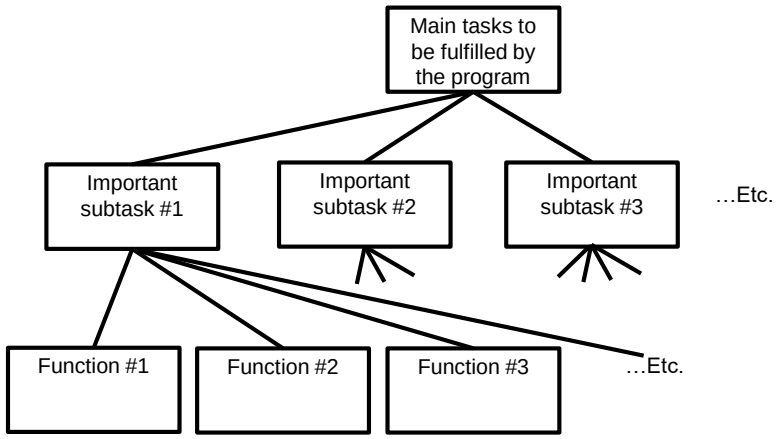
James Tam

Procedural Programming & This Course

- New terminology:
  - 'Function', 'procedure', 'subroutine' are different terms for the same programming tool (the term used depends upon the programming language).
  - The most commonly used term is 'function'.
- Functional decomposition is a key part of CPSC 217 (Exert from the university calendar description)
  - "Introduction to problem solving, analysis and design of small-scale computational systems and **implementation using a procedural programming language**."
- This is why later assignments are strict in marking – you must implement your solution using proper procedural programming techniques (taught in class).
  - Otherwise you have missed out on the major learning objective for the this course.

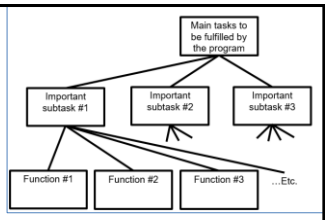
James Tam

Decomposing Your Program Into Functions According To Tasks/Features It Needs To Implement



When do you stop decomposing and start writing functions? No clear cut off but use the "Good style" principles (later in these notes) as a guide e.g., a function should have one well defined task and not exceed a screen in length.

How To Decompose A Problem Into Functions



- Break down the program by what it does (described with *actions/verbs or action phrases*).
- Eventually the different parts of the program will be implemented as functions.

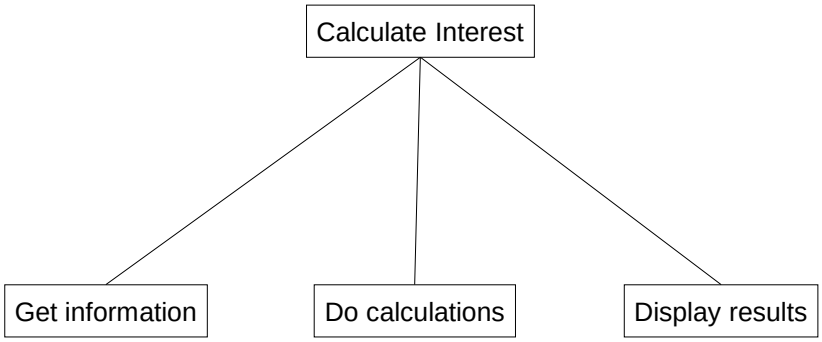
Example Problem

- Design a program that will perform a simple interest calculation.
- The program should prompt the user for the appropriate values, perform the calculation and display the values onscreen.

Example Problem

- Design a program that will perform a simple interest calculation.
- The program should *prompt* the user for the appropriate values, *perform the calculation* and *display* the values onscreen.
- Action/verb list:
  - Prompt
  - Calculate
  - Display

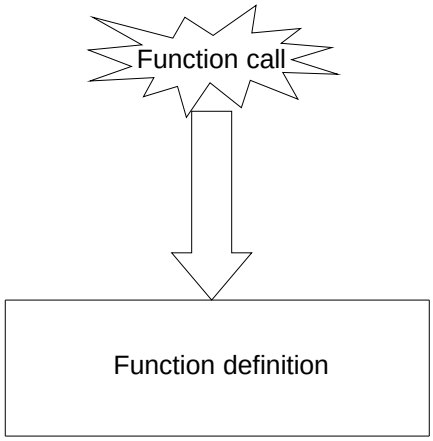
Top Down Approach: Breaking A Programming Problem Down Into Parts (Functions)



Things Needed In Order To Use Functions

- Function call
  - Actually running (executing) the function.
  - You have already done this second part many times because up to this point you have been using functions that have already been defined by someone else e.g., `print(), input()`
- Function definition
  - Instructions that indicate what the function will do when it runs.
  - Before this section: you have used built-in python functions (with their instructions already written by someone else).
  - In this section: you will learn how to write the instructions inside a function body which execute when that function runs.

Functions (Basic Case: No parameters/Inputs)



Defining A Function

- **Format:**

```
def <function name>():
    body[1]
```

- **Example:**

```
def displayInstructions():
    print ("Displaying instructions on how to use the
      program")
```

1 Body = the instruction or group of instructions that execute when the function executes (when called).

The rule in Python for specifying the body is to use indentation.

Calling A Function

- **Format:**

  `<function name>()`

- **Example**:

```
displayInstructions()
```

Quick Recap: Starting Execution Point

- The program starts at the first executable instruction that is not indented.
- In the case of your programs thus far all statement have been un-indented (save loops/branches) so it's just the first statement that is the starting execution point.

```
HUMAN_CAT_AGE_RATIO = 7
age = input("What is your age in years: ")
catAge = age * HUMAN_CAT_AGE_RATIO
…
```

- But note that the body of functions MUST be indented in Python.

James Tam

Functions: An Example That Puts Together All The Parts Of The Easiest Case

- **Name of the example program**: `1firstExampleFunction.py`
  - Learning objective:

```
def displayInstructions():
    print("Displaying instructions")

# Main body of code (starting execution point, not indented)
displayInstructions()
print("End of program")
```

```
Displaying instructions
End of program
```

James Tam

Functions: An Example That Puts Together All The Parts Of The Easiest Case

- **Name of the example program**: `1firstExampleFunction.py`

```
def displayInstructions():
    print("Displaying instructions")

# Main body of code (starting execution point)
displayInstructions()
print("End of program")
```

Function definition

Function call

James Tam

How Functions Facilitate Code Reuse

- One the function definition is complete (and tested reasonably) it can be called (reused) many times.

```
def displayInstructions():
    print("Displaying instructions")

# Main body of code (starting execution point)
displayInstructions()
displayInstructions()
displayInstructions()
```

```
========== RES
Displaying instructions
Displaying instructions
Displaying instructions
```

- Think about how many times prewritten functions such as `input` and `print` have be used.

James Tam

Defining The Main Body Of Code As A Function

- Rather than defining instructions outside of a function the main starting execution point can also be defined explicitly as a function.
- (The previous program rewritten to include an explicit start function)
  Example program: `2firstExampleFunctionV2.py`
  - Learning objective: enclosing the start of the program inside a function

```
def displayInstructions():
    print ("Displaying instructions")

def start():
    displayInstructions()
    print("End of program")
```

- **Important:** If you explicitly define the starting function then do not forgot to explicitly call it!

```
start ()
```

Don't forget to start your program! Program starts at the first executable un-indented instruction

James Tam

Stylistic Note

- By convention the starting function is frequently named '`main()`' or in my case '`start()`'.

```
def main():
```

- OR

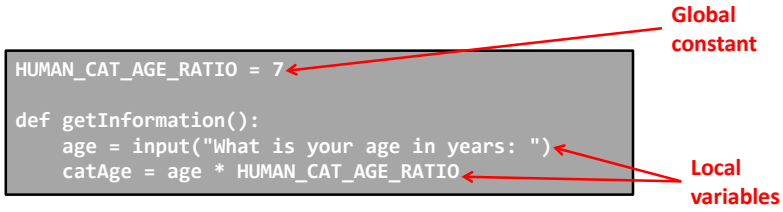
```
def start():
```

- This is done so the reader can quickly find the beginning execution point.

James Tam

New Terminology

- **Local variables**: are created within the body of a function (indented)
- **Global constants**: created outside the body of a function.
- (The significance of global vs. local is coming up shortly).



James Tam

Creating Your Variables

- Before this section of notes: all statements (including the creation of a variables) occur outside of a function

```
HUMAN_CAT_AGE_RATIO = 7
age = input("What is your age in years: ")
catAge = age * HUMAN_CAT_AGE_RATIO
…
```

- Now that you have learned how to define functions, **ALL your variables must be created with the body of a function**.
- Constants can still be created outside of a function (more on this later).

'Outside': OK for constants only

```
HUMAN_CAT_AGE_RATIO = 7

def getInformation():
    age = input("What is your age in years: ")
    catAge = age * HUMAN_CAT_AGE_RATIO
```

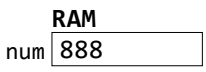
Inside function body: all variables must be here

James Tam

Reason #1: Declaring Variables Locally

- Variables are memory locations that are used for the temporary storage of information.

```
num = 888
```



- Each variable uses up a portion of memory, if the program is large then many variables may have to be declared (a lot of memory may have to be allocated to store the contents of variables).

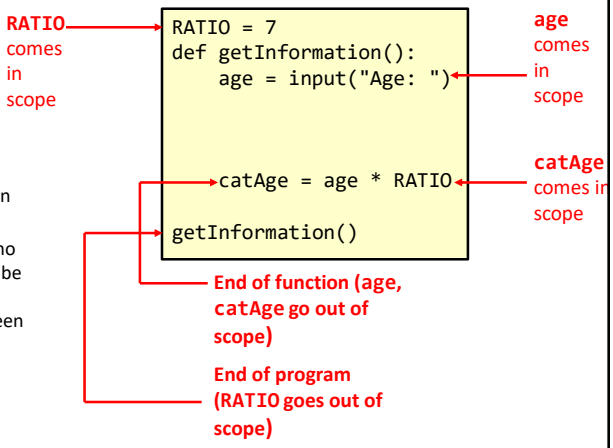
James Tam

What Is The Significance Of Being 'Local'

- To minimize the amount of memory that is used to store the contents of variables only create variables when they are needed ("allocated").
- When the memory for a variable is no longer needed it can be 'freed up' and reused ("de-allocated").
- To design a program so that memory for variables is only allocated (reserved in memory) as needed and de-allocated when they are not (the memory is free up) variables should be declared as local to a function.
- (There's an even better reason for making variables local coming up later under 'side effects').

Scope

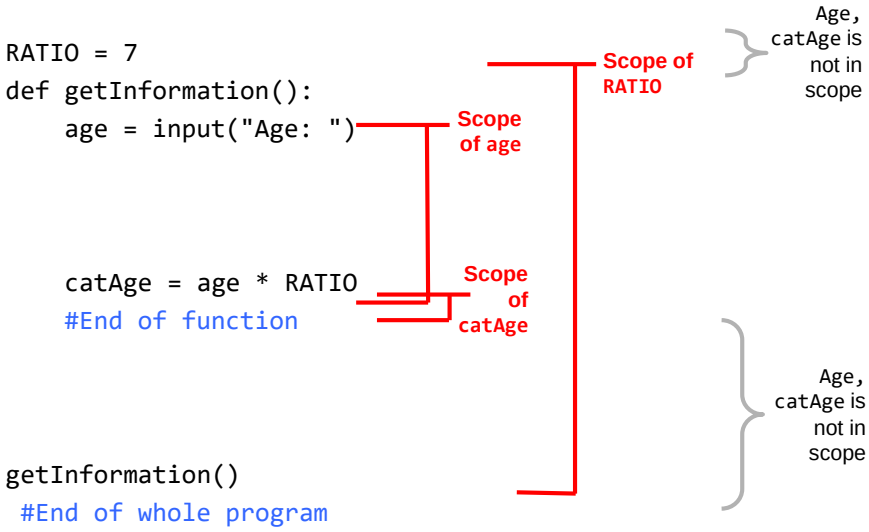
- The scope of an identifier (variable, constant) is where it may be accessed and used.
- In Python[1]:
  - An identifier comes into scope (becomes visible to the program and can be used) after it has been declared.
  - An identifier goes out of scope (no longer visible so it can no longer be used) at the end of the indented block where the identifier has been declared.



1 The concept of scoping (limited visibility) applies to all programming languages. The rules for determining when identifiers come into and go out of scope will vary with a particular language.

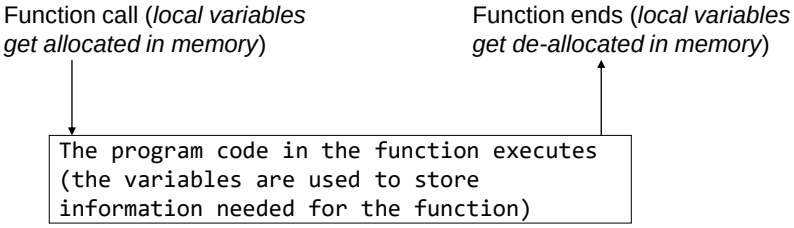
James Tam

Visually Representing Scope



James Tam

Visual Reminder Of How Locals Work



James Tam

Reminder: Where To Create Local Variables

```
def <function name>():
    Somewhere within
    the body of the
    function
    (indented part)
```

Example:

```
def fun():
    num1 = 1
    num2 = 2
```

Working With Local Variables: Putting It All Together

- **Name of the example program**: `3secondExampleFunction.py`
  - Learning objective: creating/defining variables that only exist while a function runs (local to that function).

```
def fun():
    num1 = 1
    num2 = 2
    print(num1, " ", num2)

# start function
fun()
```

Variables that are local to function 'fun'

Scope of num1

Scope of num2

```
[csc decomposition 62 ]> python secondExampleFunction.py
1   2
```

James Tam

After This Section You Should Now Know

- How and why the top down approach can be used to decompose problems
  - What is procedural programming
- How to write the definition for a function
- How to write a function call
- How and why to declare variables locally
- How to pass information to functions via parameters

James Tam

Copyright Notification

- "Unless otherwise indicated, all images in this presentation are used with permission from Microsoft."

James Tam