

# Functions: Decomposition And Code Reuse, Part 3

- Global identifiers, scope and program design
- Declaring variables: where in your function/at what level in your program
- Boolean functions
- Breaking long functions into parts
- Common errors when defining functions
- Program design and defining functions
- Testing functions
- Benefits & drawbacks of defining functions

## In Class Exercise, Functions

- Write a function called 'emphasize' that takes a string as a parameter.
- This function returns a modified version of the string:
  - !!! will be added onto the end (three exclamation marks are added to the end of the existing string).
  - Recall: The concatenation operator is the 'plus' operator '+' and it can connect two strings.

James Tam

## Declaring Variables: Stylistic Note

- Creating variables all at once at the start of a function.

```
def start():
    #Variables declared
    principle = 0
    rate = 0
    time = 0
    interest = 0
    amount = 0

    introduction()
    principle, rate, time = getInputs()
    interest, amount =
        calculate(principle, rate, time)
    display(principle, rate, time,
            interest, amount)

start()
```

**Not syntactically  
required but a  
stylistic approach**

Origins: many languages (e.g. C, C++, Java, Pascal) require variables to be declared with a specific type before they can be used:

```
fun ()
{
    //Variables declared
    Scanner in = null;
    int age = 0;

    in = new Scanner(System.in);
    age = in.nextInt();
    System.out.print("Age:");
}
```

James Tam

## Global Scope (Again)

- Identifiers (constants or variables) that are declared within the body of a function have a local scope (the function).

```
def fun():
    num = 12
    # End of function fun
```

**Scope of num is the function**

- Identifiers (constants or variables) that are created outside the body of a function have a global scope (the program).

```
num = 12
def fun1():
    # Instructions

def fun2():
    # Instructions

# End of program
```

**Scope of num is the entire program**

James Tam

## Global Scope: An Example

- **Name of the example program:** 7simple\_global\_example.py
  - Learning objective: how global variables are accessible throughout a program.

```
num1 = 10
```

```
def fun():
    print(num1) 10
```

```
def start():
    fun()
    print(num2) 20
```

```
num2 = 20
```

```
start()
```

James Tam

## Global Variables: General Characteristics

- You can access the contents of global variables anywhere in the program.
  - Python: this can occur even if the 'global' keyword is not used.
- In most programming languages you can also modify global variables anywhere as well.
  - This is why the usage of global variables is regarded as bad programming style, they can be accidentally modified anywhere in the program.
  - Changes in one part of the program can introduce unexpected side effects in another part of the program.
  - So unless you have a compelling reason you should NOT be using global variables but instead you should pass variables as parameters/returning values.
    - Unless you are told otherwise using global variables can affect the style component of your assignment grade.
    - Global constants are acceptable and are commonly used.

James Tam

## Global Variables: Python Specific Characteristic

- **Name of the example program:** 8globals\_vs\_locals.py
  - Learning objective: Relationship between accessing global variables and creating locals.

```
num = 1
def fun():
    num = 2 2 Local created and displayed
    print(num)
def start():
    print(num) 1 Global
    fun()
    print(num) 1 Global

start()
```

James Tam

## Scoping Rules: Globals

- When an identifier is referenced (variable or constant) then:
  1. First look in the local scope for the creation of the identifier: if found here then stop looking and use this identifier
  2. If nothing exists at the local level then look globally

num = <value> here?

```
def aFunction():
    num = <value> here?
    print(num)
```

Reference to  
an identifier

James Tam

## Python Globals: 'Read' But Not 'Write' Access

- By default global variables can be accessed globally (read access).
- Attempting to change the value of global variable will only create a new local variable by the same name (no write access to the global, a local is created).

```
num = 1
```

← **Global num**

```
def fun():
    num = 2
    print(num)
```

← **Local num**

- Prefacing the name of a variable with the keyword 'global' in a function will indicate changes in the function will refer to the global variable rather than creating a local one.

```
global <variable name>
```

James Tam

## Globals: Another Example ('Write' Access Via The "Global" Keyword)

- **Name of the example program:** 9modifying\_globals.py

— Learning objective: How global variables can be modified inside functions.

```
num = 1
```

```
def fun():
```

```
    global num
```

```
    num = 2
```

```
    print(num)
```

References to the name 'num' now affect the global variable, local variable not created inside function 'fun'

2

Global changed

```
def start():
```

```
    print(num)
```

```
    fun()
```

```
    print(num)
```

1

Global

2

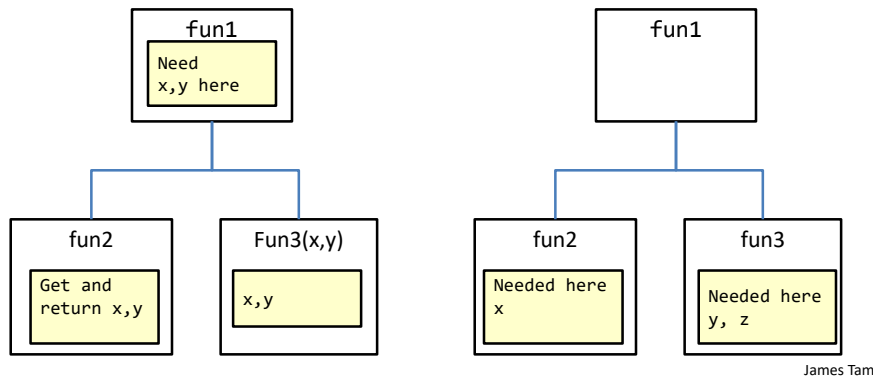
Global still changed after 'fun()' is done

```
start()
```

James Tam

## What Level To Declare Variables

- Declare your variables as local to a function.
- When there are multiple levels of functions (a level is formed when one function calls another) then:
  - A variable should be created at the lowest level possible



James Tam

## Boolean Functions

- Return a Boolean value (true/false): “Asks a question”
- Typically the Boolean function will ‘ask the question’ about a parameter(s)
- Example:
  - Is it true that the string can be converted to a number?

```

aString = input("Enter age: ")
ageOK = isNum(aString)
if (ageOK != True):
    print("Age must be a numeric value")
else:
    # OK to convert the string to a number
    age = int(aString)

```

**# Boolean function**  
**def isNum(aString):**  
**# Returns (True**  
**# or False)**

A red arrow points from the `isNum(aString)` call in the code to the function definition on the right.

James Tam

## Example: How To Decompose A Long Function

- To decompose (break into parts) long functions examine the structure for sections e.g. loops (and their bodies), branches (and their bodies).
- Each of these sections may be a candidate to be moved into it's own separate function body:

**Before**

```
def fun1():
    while(BE1):
        if(BE2):
            #If body #1
        if(BE3):
            #If body #2
```

**After**

```
def fun3():
    #If body #2

def fun2():
    #If body #1

def fun1():
    while(BE1):
        if(BE2):
            fun2()
        if(BE3):
            fun3()
```

James Tam

## Functions Should Be Defined Before They Can Be Called!

### • Correct 😊

```
def fun():
    print("Works")
```

} **Function definition**

# Start  
fun() } **Function call**

### • Incorrect ☹️

```
# Start  
fun() }
```

} **Function call**

```
def fun():
    print("Doesn't work")
```

} **Function definition**

## Another Common Mistake

- Forgetting the brackets during the function call:

```
def fun():  
    print("In fun")  
  
# Start of program  
print("Starting the program")  
fun
```

James Tam

## Another Common Mistake

- Forgetting the brackets during the function call:

```
def fun():  
    print("In fun")  
  
# Start of program  
print("Program started")  
fun()
```

With python the missing set  
of brackets do not produce a  
syntax/translation error

James Tam

## Another Common Problem: Indentation

- Recall: In Python indentation indicates that statements are part of the body of a function.
- (In other programming languages the indentation is not a mandatory part of the language but indenting is considered good style because it makes the program easier to read).

- Forgetting to indent:

```
def start():  
    print("start")
```

```
start()
```

James Tam

## Another Common Problem: Indentation (2)

- Inconsistent indentation:

```
def start():  
    print("first")  
    # Error: Unless this is the body of branch or loop  
    print("second")
```

```
start()
```

James Tam

## Creating A Large Document

- Recall: When creating a large document you should plan out the parts before doing any actual writing.

### Step 1: Outline all the parts (no writing)

#### Chapter 1

- Introduction
- Section 1.1
- Section 1.2
- Section 1.3
- Conclusion

#### Chapter 2

- Introduction
- Section 2.1
- Section 2.2
- Section 2.3
- Section 2.4
- Conclusion

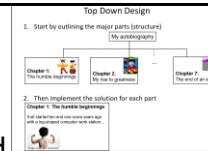
#### Chapter 3

- Introduction
- Section 3.1
- Section 3.2
- Conclusion

### Step 2: After all parts outlined, now commence writing one part at a time

#### Section 1.1

It all started seven  
and two score  
years ago...



James Tam

## Creating A Large Program

- When writing a large program you should plan out the parts before doing any actual writing.

### Step 1: Calculate interest (write empty 'skeleton' functions)

```
def getInformation():    def doCalculations():    def displayResults():
```

### Step 2: All functions outlined, write function bodies one-at-a-time (test before writing next function)

```
def getInformation():
    principle = int(input())
    interest = int(input())
    time = int(input())
    return(principle,interest,time)
```

```
# Simple test: check inputs
# are properly read as input
# and returned to caller
p,r,t = getInformation()
print(p,r,t)
```

James Tam

## Yet Another Problem: Creating 'Empty' Functions

```
def start():
```

```
start()
```

**Problem:** This statement appears to be a part of the body of the function but it is not indented???!?

James Tam

## **Solution** When Outlining Your Program By Starting With 'Empty' Functions

```
def fun():
```

```
    print()
```

A function must have at least one instruction in the body

```
# Program's start
```

```
fun()
```

Alternative (writing an empty function: 'pass' a python instruction that literally does nothing)

```
def fun():
```

```
    pass
```

```
# Program's start
```

```
fun()
```

James Tam

## Testing Functions

- The correctness of a function should be verified. (“Does it do what it is supposed to do?”)
- Typically this is done by calling the function, passing in predetermined parameters and checking the result.
- Example: 10absolute\_test.py

```
def absolute(number):
    if (number < 0):
        result = number * -1
    else:
        result = number
    return(result)
```

### # Test cases

```
print(absolute(-13))
print(absolute(7))
```

Expected results:

13  
7

James Tam

## Why Employ Problem Decomposition And Modular Design (1)

- Drawback
  - Complexity – understanding and setting up inter-function communication may appear daunting at first.
  - Tracing the program may appear harder as execution appears to “jump” around between functions.
  - These are ‘one time’ costs: once you learn the basic principles of functions with one language then most languages will be similar.

## Why Employ Problem Decomposition And Modular Design (2)

- Benefit
  - Solution is easier to visualize and create (decompose the problem so only one part of a time must be dealt with).
  - Easier to test the program:
    - Test one feature/function at a time
    - (Testing multiple features increases complexity)
  - Easier to maintain (if functions are independent changes in one function can have a minimal impact on other functions, if the code for a function is used multiple times then updates only have to be made once).
  - Less redundancy, smaller program size (especially if the function is used many times throughout the program).
  - Smaller programs size: if the function is called many times rather than repeating the same code, the function need only be defined once and then can be called many times.

James Tam

## After This Section You Should Now Know

- What is global scope
- Consequences of employing global scope
- What are scoping rules when referring to an identifier
- Where variables should be declared in the body of a function
- A guideline for the level at which variables should be declared
- How/when to employ doc string documentation
- What is a Boolean function
- A technique for decomposing a long function into smaller functions
- Common errors when defining functions
- The basics of testing a function
- The benefits & drawbacks of defining functions

James Tam

## Copyright Notification

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

James Tam