

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

Writing Your Own Functions: Why Do It?

- **First reason, you have no choice:** the code hasn't been implemented for this feature yet.
- Example: you can't just look up the prebuilt functions in python and have one of them do all the work for one of your assignments.

James Tam

Writing Your Own Functions: Why Do It?

- **Second reason, you need to know this:** it's not only done all the time in real life but it's a key component of this course.
- (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.** ”
 - **Expectation students who have successfully finished this course will be able to properly implement a non-trivial program not only using functional decomposition but also apply important related concepts such as: parameters, return values and scope.**
 - This is why later assignments are strict in marking – you must implement your solution using proper procedural programming techniques (taught in class).
- **New terminology:**
 - Function, procedure, method
 - For now you can think of them as largely interchangeable although you will learn the difference between a function and method towards the end of this course.
 - Most languages don't distinguish procedures from functions.

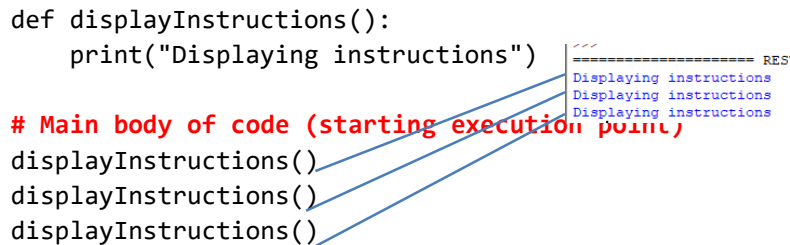
James Tam

Writing Your Own Functions: Why Do It?

- **Third reason, reuse/efficiency:** Once 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()
```



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

James Tam

Writing Your Own Functions: Why Do It?

- **Fourth reason, easier maintenance:** (related to the previous benefit: write once, use many times): when program maintenance (changes to code) is needed.
- If the same code is written over and over again in different parts of the program then each location must be changed.
- Implementing that same code in one function requires only changes to the code in that function.

```
def myFunction():
    #Just modify here
```

```
#Version: no functions requires
#many modifications
#Code to modify

#Code to modify

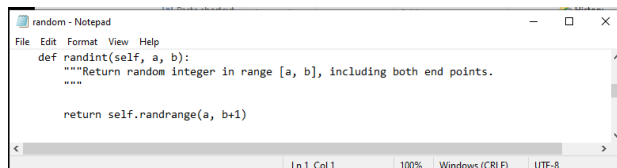
#Code to modify
```

- This may result in a smaller program with fewer/no redundancies as well.

James Tam

Writing Your Own Functions: Why Do It?

- **Fifth reason, decoupling of your code:**
- New terminology, decoupling: a fancy term for a simple concept.
- In this case it means you can simply use a function without worrying about the 'internal' details of how it was written.
- You simply need things such as: how to call it, what operations the function implements, what are its return values etc.
- This is the actual code from the `randint()` function.
 - You just have to know how to call it not know all the intimate details of how every line works.



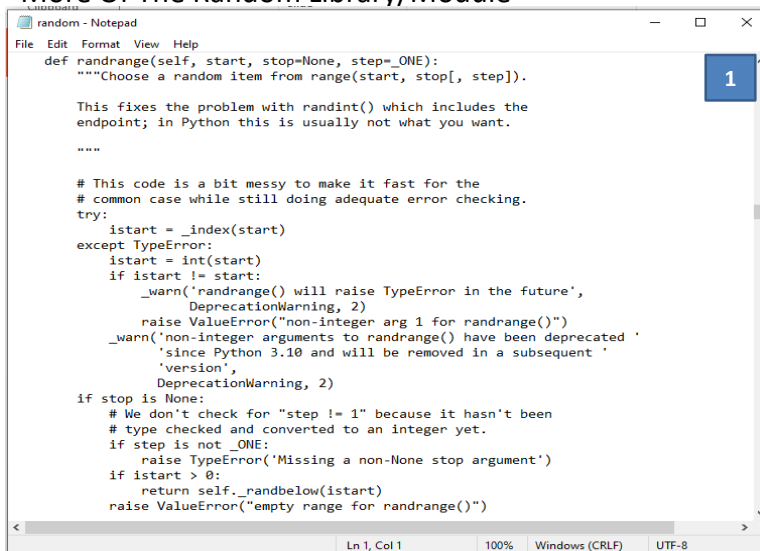
```

random - Notepad
File Edit Format View Help
def randint(self, a, b):
    """Return random integer in range [a, b], including both end points.
    """
    return self.randrange(a, b+1)
  
```

James Tam

Writing Your Own Functions: Why Do It?

- More Of The Random Library/Module



```

random - Notepad
File Edit Format View Help
def randrange(self, start, stop=None, step=_ONE):
    """Choose a random item from range(start, stop[, step]).

    This fixes the problem with randint() which includes the
    endpoint; in Python this is usually not what you want.

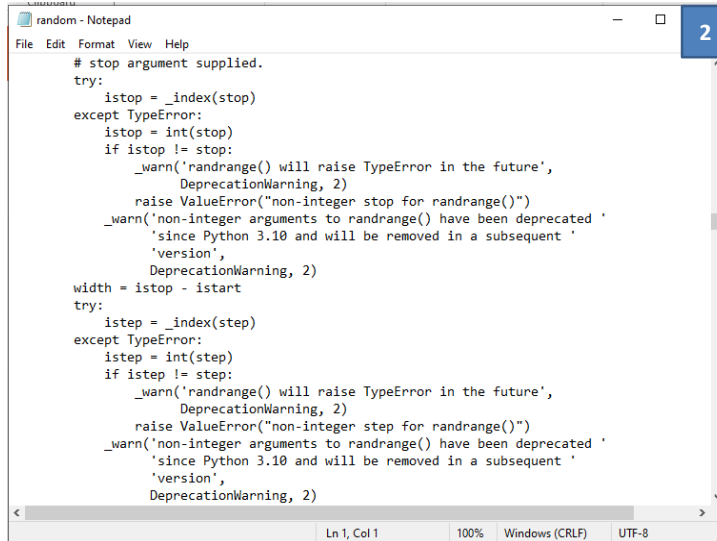
    """

    # This code is a bit messy to make it fast for the
    # common case while still doing adequate error checking.
    try:
        istart = _index(start)
    except TypeError:
        istart = int(start)
        if istart != start:
            _warn('randrange() will raise TypeError in the future',
                  DeprecationWarning, 2)
            raise ValueError("non-integer arg 1 for randrange()")
        _warn('non-integer arguments to randrange() have been deprecated '
              'since Python 3.10 and will be removed in a subsequent '
              'version',
              DeprecationWarning, 2)
    if stop is None:
        # We don't check for "step != 1" because it hasn't been
        # type checked and converted to an integer yet.
        if step is not _ONE:
            raise TypeError('Missing a non-None stop argument')
        if istart > 0:
            return self._randbelow(istart)
        raise ValueError("empty range for randrange()")
  
```

James Tam

Writing Your Own Functions: Why Do It?

- More Of The Random Library/Module



```

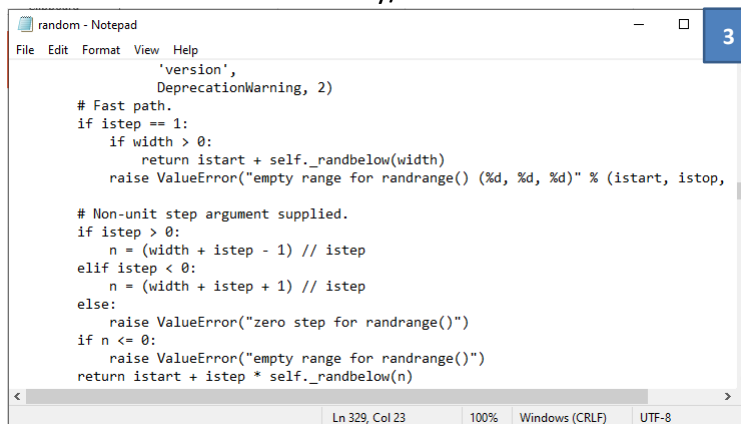
random - Notepad
File Edit Format View Help
# stop argument supplied.
try:
    istop = _index(stop)
except TypeError:
    istop = int(stop)
    if istop != stop:
        _warn('randrange() will raise TypeError in the future',
              DeprecationWarning, 2)
        raise ValueError("non-integer stop for randrange()")
    _warn('non-integer arguments to randrange() have been deprecated '
          'since Python 3.10 and will be removed in a subsequent '
          'version',
          DeprecationWarning, 2)
width = istop - istart
try:
    istep = _index(step)
except TypeError:
    istep = int(step)
    if istep != step:
        _warn('randrange() will raise TypeError in the future',
              DeprecationWarning, 2)
        raise ValueError("non-integer step for randrange()")
    _warn('non-integer arguments to randrange() have been deprecated '
          'since Python 3.10 and will be removed in a subsequent '
          'version',
          DeprecationWarning, 2)
Ln 1, Col 1 100% Windows (CRLF) UTF-8

```

James Tam

Writing Your Own Functions: Why Do It?

- More Of The Random Library/Module



```

random - Notepad
File Edit Format View Help
    'version',
    DeprecationWarning, 2)
# Fast path.
if istep == 1:
    if width > 0:
        return istart + self._randbelow(width)
    raise ValueError("empty range for randrange() (%d, %d, %d)" % (istart, istop,
# Non-unit step argument supplied.
if istep > 0:
    n = (width + istep - 1) // istep
elif istep < 0:
    n = (width + istep + 1) // istep
else:
    raise ValueError("zero step for randrange()")
if n <= 0:
    raise ValueError("empty range for randrange()")
return istart + istep * self._randbelow(n)
Ln 329, Col 23 100% Windows (CRLF) UTF-8

```

James Tam

Writing Your Own Functions: Why Do It?

- **Sixth reason:** to simplify the problem.
- 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

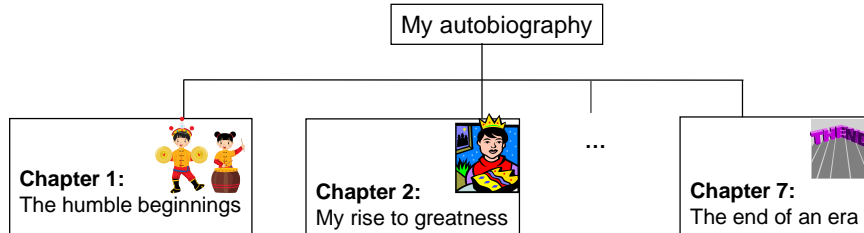
Solving Larger Problems

- 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

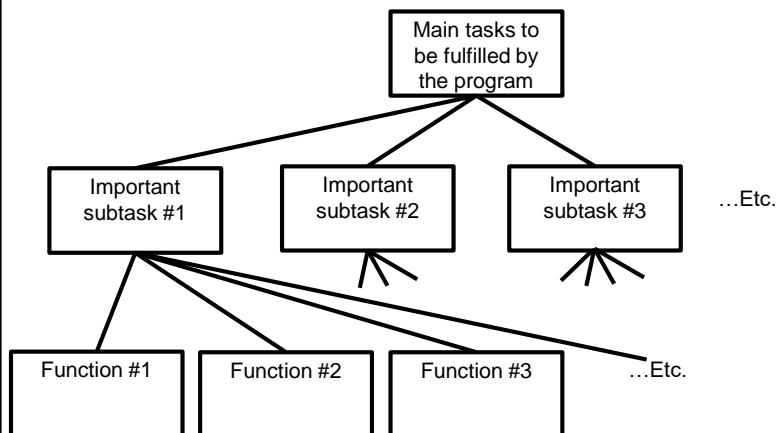
Chapter 1: The humble beginnings

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



Image copyright unknown

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



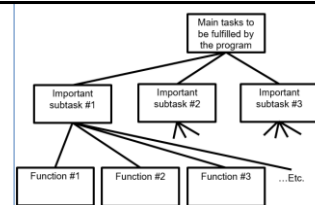
When do you stop decomposing and start writing functions? No clear cut off but use the "Good style" principles (later in these notes) as a guide e.g., a function should have one well defined task and not exceed a screen in length.

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

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.

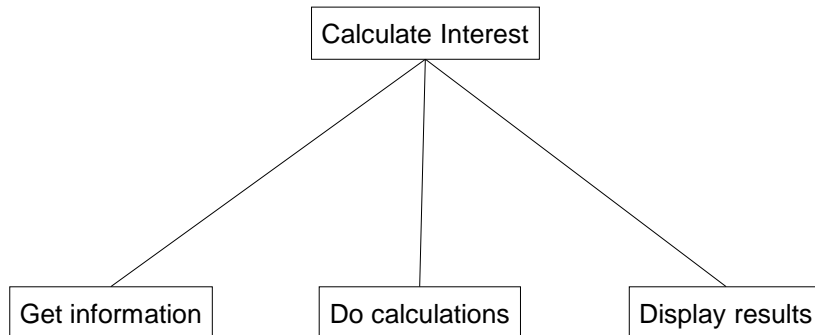
Example Problem

- 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

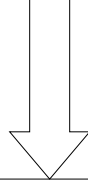
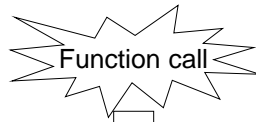
Top Down Approach: Breaking A Programming Problem Down Into Parts (Functions)



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)



You've already called prebuilt functions and passed no arguments e.g. `print()`, `input()`

Function definition

Defining A Function

- **Format:**

```
def <function name>¹():
    body²
```

- **Example:**

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

- You don't need to define prebuilt functions because some else has defined the code for you.

```
def randint(self, a, b):
    """Return random integer in range [a, b], including both end points.
    """
    return self.randrange(a, b+1)
```

1 Functions should be named according to the rules for naming variables (all lower case alphabetic, separate multiple words via camel case or by using an underscore).

2 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()`

- As you mentioned you have already learned how to call a prewritten function e.g. `print()`, `int()`, `input()`, `randint(1,6)` etc.

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 statements 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")
```

(Something new
in this section):
Function
definition

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

(You've done
this before):
Function call

James Tam

Defining The Main Body Of Code As A Function

- Good style: unless it's mandatory, all instructions must be inside 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 forget 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).

```
HUMAN_CAT_AGE_RATIO = 7

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

Global
constant

Local
variables

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).

```
HUMAN_CAT_AGE_RATIO = 7

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

'Outside': OK for
constants only

Inside function
body: all variables
(e.g. 'age',
'catAge') must be
here

James Tam

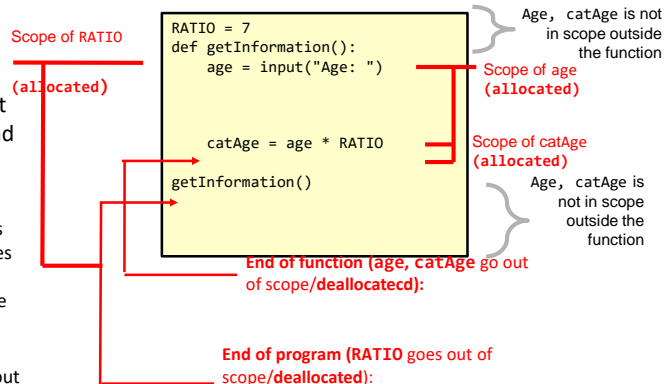
Local Variables

- Characteristics
 - Locals only get allocated (created in memory) when the function is called.
 - Locals get de-allocated (unavailable in memory) when the function ends.
- Benefits (why create them this way)
 - 1st: more efficient use of memory
 - 2nd: minimize the occurrence of side effects of global variables
 - This is the main reason why it's regarded as bad style in actual practice.
 - But details are more complex so the explanation will come later.
 - 3rd: pedagogical (creating variables locally forces you to apply important programming concepts such as parameter passing, function return values and scope).

James Tam

Scope: Visually Showing When Memory Locations Can Be Accessed

- The scope of an identifier (variable, constant) is where it may be accessed and used.
- In Python¹:
 - 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

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)
```

Variables that are local to function 'fun'

Scope of num1

Scope of num2

start function

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

James Tam

Variables Vs. Named Constants

- As you have already been taught:
 - Variables can change as the programs run while named constants don't change after they've been set to the initial value.
 - To visually distinguish the two variables use lower case while constants are capitalized.
- Your program should consistently distinguish the two!
 - The following is only a 'constant' in name only and is treated like a variable.

```
PI = 3.14
radius = 10
area = PI * (radius ** 2)
```

PI = 3.1 #Do not change the value in a constant!

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