Return Values

SECTION 5.4

Return Values

- Functions can (optionally) return one value
  - Add a return statement that returns a value
    - A return statement does two things:
      1) Immediately terminates the function
      2) Passes the return value back to the calling function

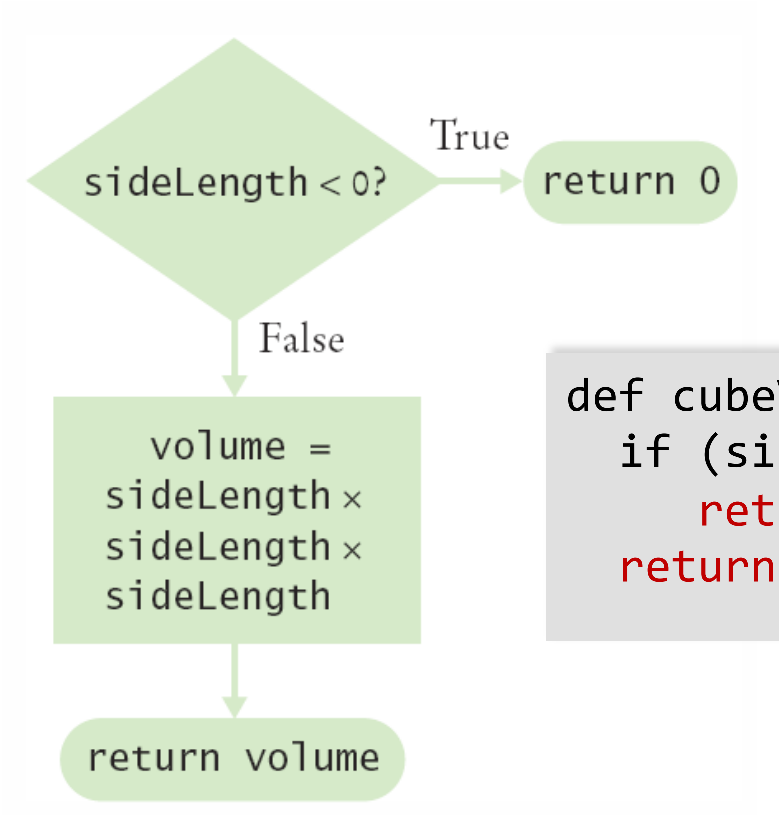
```
def cubeVolume (sideLength):
   volume = sideLength * 3
   return volume
```

return statement

The return value may be a value, a variable or a calculation

Multiple `return` Statements

- A function can use multiple `return` statements
  - But every branch must have a `return` statement



```
def cubeVolume(sideLength):
  if (sideLength < 0):
     return 0
  return sideLength * 3
```

Multiple `return` Statements (2)

- Alternative to multiple returns (e.g., one for each branch):
  - You can avoid multiple returns by storing the function result in a variable that you return in the last statement of the function
  - For example:

```
def cubeVolume(sideLength) :
    if sideLength >= 0:
        volume = sideLength ** 3
    else :
        volume = 0
    return volume
```

Make Sure A Return Catches All Cases

- Missing return statement
  - Make sure all conditions are handled
  - In this case, `sideLength` could be equal to 0
    - No return statement for this condition
    - The compiler will *not* complain if any branch has no return statement
    - It may result in a run-time error because Python returns the special value **None** when you forget to return a value

```
def cubeVolume(sideLength) :
    if sideLength >= 0 :
        return sideLength ** 3
    # Error—no return value if sideLength < 0
```

Make Sure A Return Catches All Cases (2)

- A correct implementation:

```
def cubeVolume(sideLength) :
    if sideLength >= 0
        return sideLength ** 3
    else :
        return 0
```

Implementing a Function: Steps

1. Describe what the function should do
   i. Provide a simple "liberal arts terms" description of what the functions does
   ii. "Compute the volume of a pyramid with a square base"
2. Determine a list of all of the functions inputs
   i. Make a list of ***all*** of the parameters that can vary
   ii. Do not be overly specific
3. Determine the types of the parameter variables and the return value

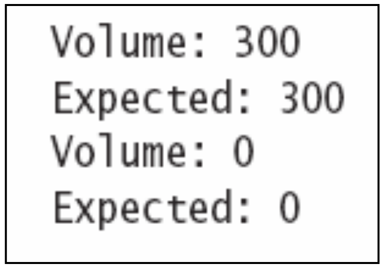
Implementing a Function: Steps

4) Write pseudocode for obtaining the desired result
   i. Express an mathematical formulas, branches and loops in pseudocode
5) Implement the function body

```
def pyramidVolume(height, baseLength) :
    baseArea = baseLength * baseLength
    return height * baseArea / 3
```

Implementing a Function: Steps

6) Test your function
    i. Design test cases and code



```
Volume: 300
Expected: 300
Volume: 0
Expected: 0
```

Pyramids.py

- Open the file pyramids.py
- Look at how the main function is set up to make the calls to `pyramidVolume` and print the expected results

Functions Without Return Values

SECTION 5.5

Functions Without Return Values

- functions are not required to return a value
  - No return statement is required
  - The function can generate output even when it doesn’t have a return value

```
. . .
boxString("Hello")
. . .
```

```
-------
!Hello!
-------
```

```
def boxString(contents) :
    n = len(contents) :
    print("-" * (n + 2))
    print("!" + contents + "!")
    print("-" * (n + 2))
```

Using `return` Without a Value

- You can use the return statement without a value
  - The function will terminate immediately!

```
def boxString(contents) :
    n = len(contents)
    if n == 0 :
        return # Return immediately
    print("-" * (n + 2))
    print("!" + contents + "!")
    print("-" * (n + 2))
```

Reusable Functions

SECTION 5.6

Problem Solving: Reusable Functions

- **Find repetitive code**
  - May have different values but same logic

```
hours = int(input("Enter a value between 0 and 23: "))
while hours < 0 or hours > 23 :
    print("Error: value out of range.")
    hours = int(input("Enter a value between 0 and 23: "))


minutes = int(input("Enter a value between 0 and 59: "))
while minutes < 0 or minutes > 59 :
    print("Error: value out of range.")
    minutes = int(input("Enter a value between 0 and 59: "))
```

0 - 23

0 - 59

Write a ‘Parameterized’ Function

```
## Prompts a user to enter a value up to a given maximum until the user
provides
# a valid input.
# @param high an integer indicating the largest allowable input
# @return the integer value provided by the user (between 0 and high,
inclusive)
#
def readIntUpTo(high) :
    value = int(input("Enter a value between 0 and " + str(high) + ":
"))
    while value < 0 or value > high :
        print("Error: value out of range.")
    value = int(input("Enter a value between 0 and " + str(high) + ":
"))

    return value
```

Readtime.py

- Open the file readtime.py
- Test the program with several inputs
    - How would you modify your project to use the readInBetween function?

An Alternate If Structure

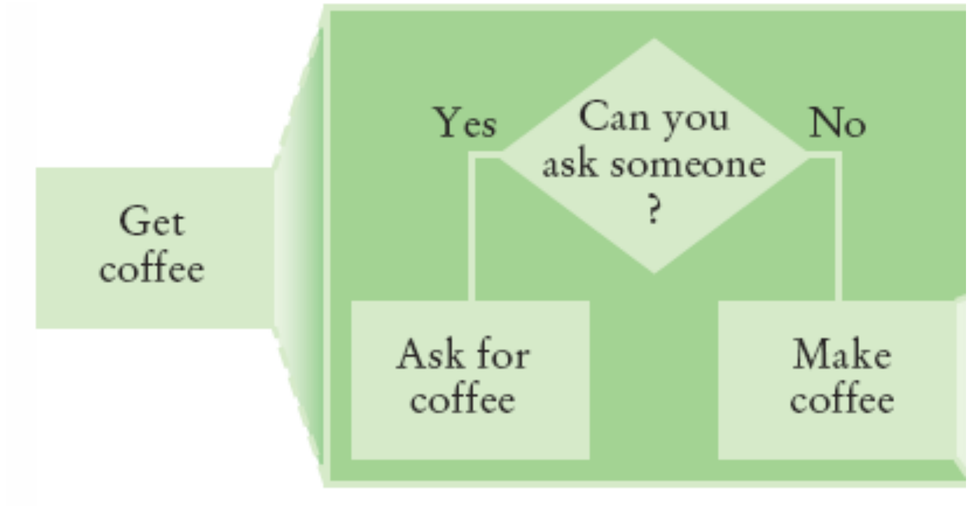
- Open the file earthquake.py
- The file contains two functions that solve the Richter scale problem from earlier this semester
  - The first uses an “`if – elif`” construct
  - The second uses single-line compound statements (Special Topic 5.1, p. 256)
  - This form of an `if` statement is very useful in functions that select and return a single value from a set of values

Stepwise Refinement

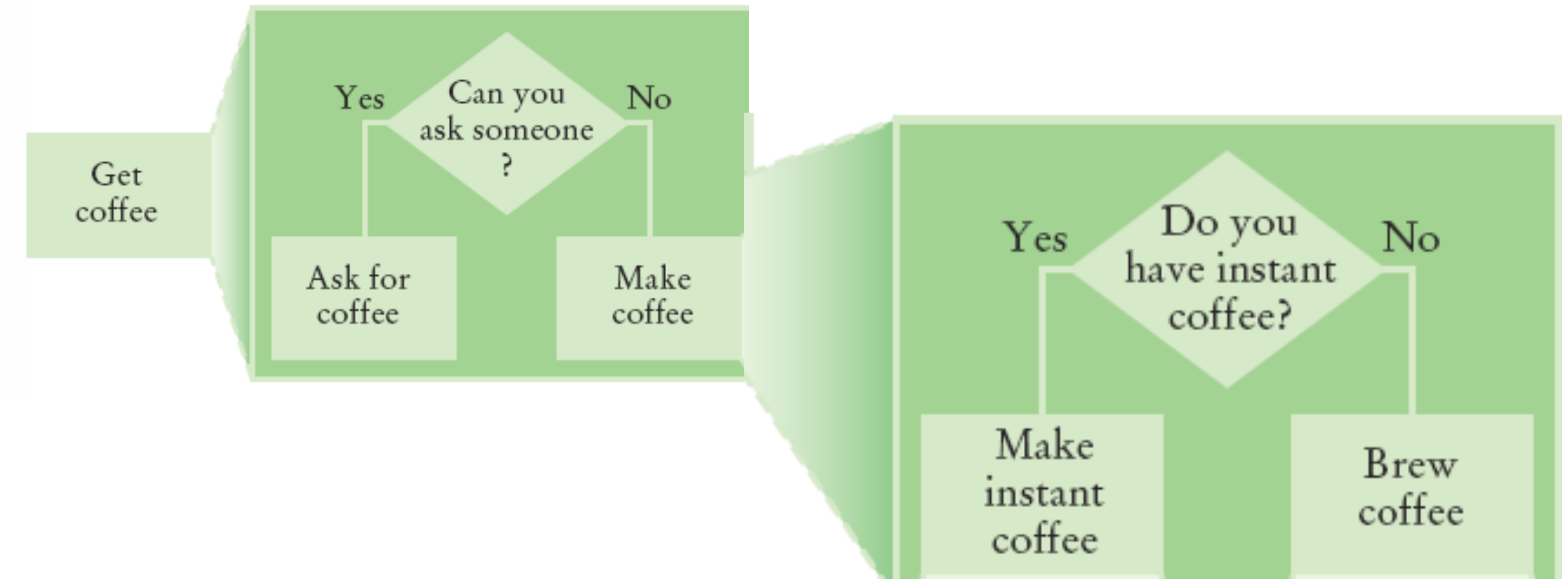
SECTION 5.7

Stepwise Refinement

- To solve a difficult task, break it down into simpler tasks
- Then keep breaking down the simpler tasks into even simpler ones, until you are left with tasks that you know how to solve



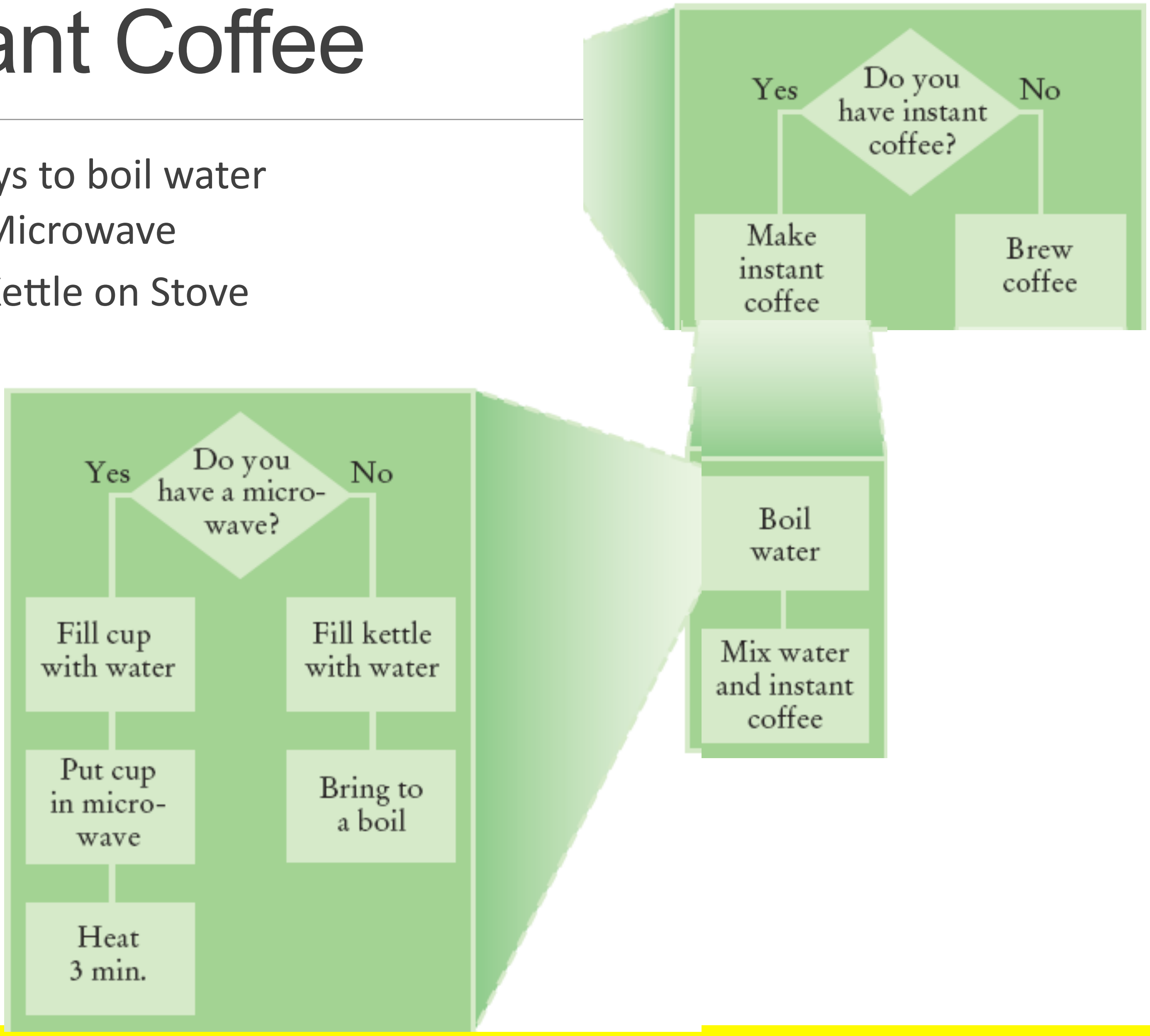
Get Coffee



- If you must make coffee, there are two ways:
  - Make Instant Coffee
  - Brew Coffee

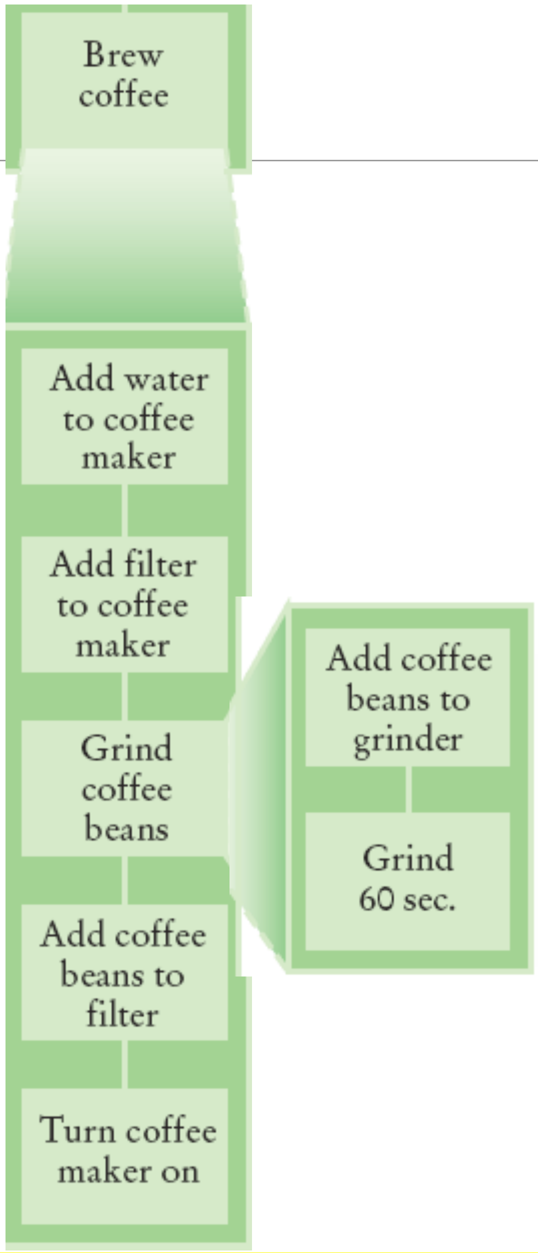
Instant Coffee

- Two ways to boil water
  1) Use Microwave
  2) Use Kettle on Stove



Brew Coffee

- Assumes coffee maker
  - Add water
  - Add filter
  - Grind Coffee
    - Add beans to grinder
    - Grind 60 seconds
  - Fill filter with ground coffee
  - Turn coffee maker on
- Steps are easily done


Brew coffee
Add water to coffee maker
Add filter to coffee maker
Grind coffee beans
Add coffee beans to filter
Turn coffee maker on
Add coffee beans to grinder
Grind 60 sec.

Stepwise Refinement Example

- When printing a check, it is customary to write the check amount both as a number ("$274.15") and as a text string ("two hundred seventy four dollars and 15 cents")
- Write a program to turn a number into a text string
- Wow, sounds difficult!
- Break it down
  - Let's take the dollar part (274) and come up with a plan
  - Take an Integer from 0 – 999
  - Return a String
  - Still pretty hard…

Stepwise Refinement Example

- Take it digit by digit (2, 7, 4) – left to right
- Handle the first digit (hundreds)
  - If empty, we are done with hundreds
  - Get first digit (Integer from 1 – 9)
  - Get digit name (“one”, “two”, “three”…)
  - Add the word “hundred”
  - Sounds easy!
- Second digit (tens)
  - Get second digit (Integer from 0 – 9)
  - If 0, we are done with tens… handle third digit
  - If 1, … may be eleven, twelve... Teens… Not easy!
    - Let’s look at each possibility left (1x-9x)…

Stepwise Refinement Example

- If second digit is a 0
  - Get third digit (Integer from 0 – 9)
  - Get digit name (“”, “one”, “two”…) … Same as before?
  - Sounds easy!
- If second digit is a 1
  - Get third digit (Integer from 0 – 9)
  - Return a String (“ten”, “eleven”, “twelve”…)
- If second digit is a 2-9
  - Start with string “twenty”, “thirty”, “forty”…
  - Get third digit (Integer from 0 – 9)
  - Get digit name (“”, “one”, “two”…)   … Same as before
  - Sounds easy!

Name the Sub-Tasks

- digitName
  - Takes an Integer from 0 – 9
  - Return a String ("", "one", "two"…)
- tensName (second digit >= 20)
  - Takes an Integer from 0 – 9
  - Return a String ("twenty", "thirty"…) plus
    - digitName(third digit)
- teenName
  - Takes an Integer from 0 – 9
  - Return a String ("ten", "eleven"…)

Write Pseudocode

```
part = number (The part that still needs to be converted)
name = “” (The name of the number)
If part >= 100
  name = name of hundreds in part + " hundred"
  Remove hundreds from part
If part >= 20
  Append tensName(part) to name
  Remove tens from part
Else if part >= 10
  Append teenName(part) to name
  part = 0
If (part > 0)
  Append digitName(part) to name
```

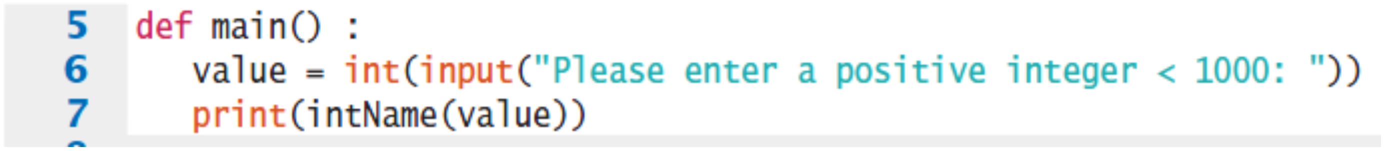
Identify functions that we can use (or re-use!) to do the work

Plan The Functions

- Decide on name, parameter(s) and types and return type
- def intName (number):
    - Turns a number into its English name
    - Returns a String that is the English description of a number (e.g., “seven hundred twenty nine”)
- def digitName (digit):
    - Return a String (“”, “one”, “two”…)
- def tensName (number):
    - Return a String (“twenty”, “thirty”…) plus
        - Return from digitName(thirdDigit)
- def teenName (number):
    - Return a String (“ten”, “eleven”…)

Convert to Python: intName Function

- Open the file intname.py in Wing
- main calls intName
  - Does all the work
  - Returns a String
- Uses functions:
  - tensName
  - teenName

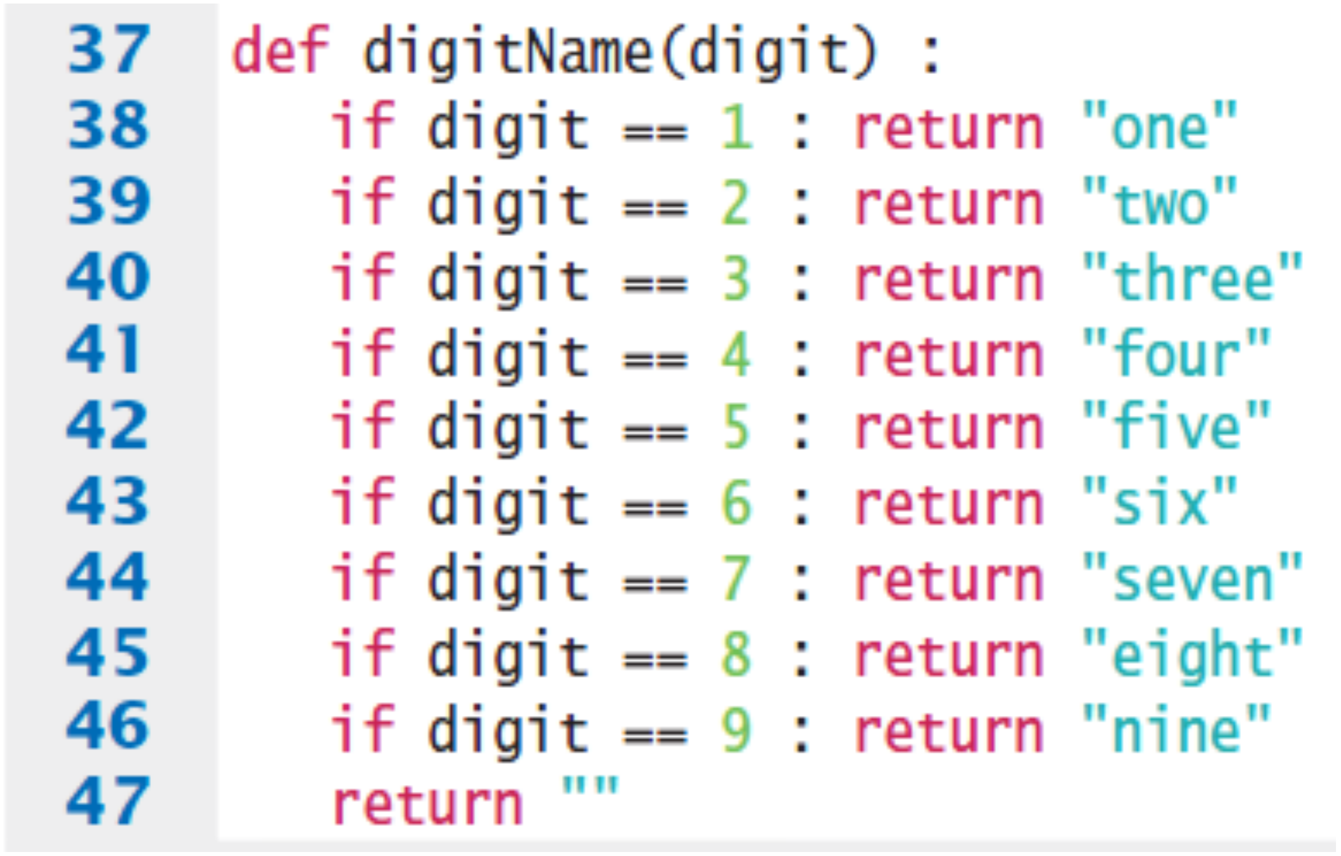


```
def main() :
   value = int(input("Please enter a positive integer < 1000: "))
   print(intName(value))
```

intName

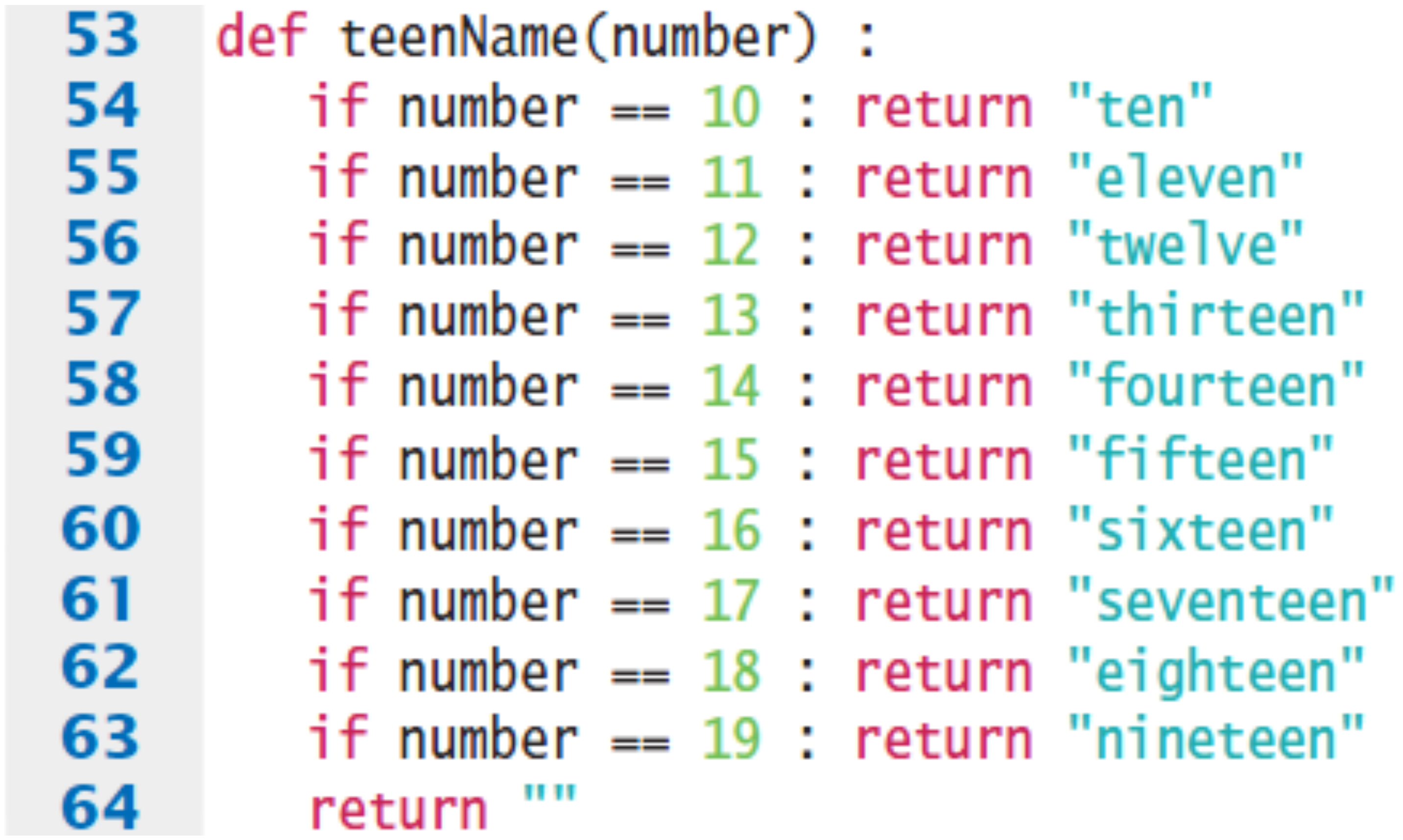
```
def intName(number) :
    part = number    # The part that still needs to be converted.
    name = ""    # The name of the number.

    if part >= 100 :
        name = digitName(part // 100) + " hundred"
        part = part % 100

    if part >= 20 :
        name = name + " " + tensName(part)
        part = part % 10
    elif part >= 10 :
        name = name + " " + teenName(part)
        part = 0

    if part > 0 :
        name = name + " " + digitName(part)

    return name
```

digitName



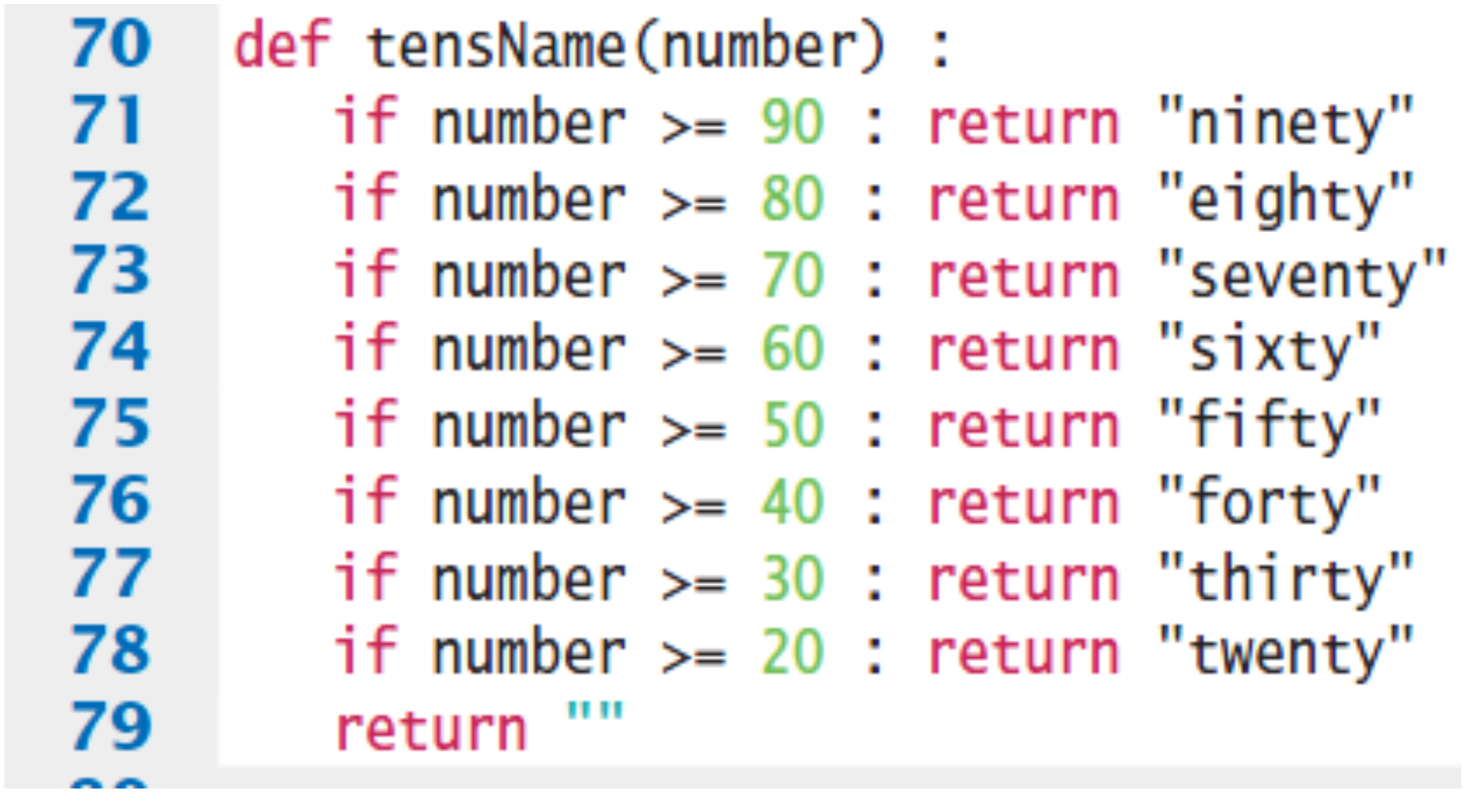
```
def digitName(digit) :
    if digit == 1 : return "one"
    if digit == 2 : return "two"
    if digit == 3 : return "three"
    if digit == 4 : return "four"
    if digit == 5 : return "five"
    if digit == 6 : return "six"
    if digit == 7 : return "seven"
    if digit == 8 : return "eight"
    if digit == 9 : return "nine"
    return ""
```

teenName



```python
def teenName(number) :
    if number == 10 : return "ten"
    if number == 11 : return "eleven"
    if number == 12 : return "twelve"
    if number == 13 : return "thirteen"
    if number == 14 : return "fourteen"
    if number == 15 : return "fifteen"
    if number == 16 : return "sixteen"
    if number == 17 : return "seventeen"
    if number == 18 : return "eighteen"
    if number == 19 : return "nineteen"
    return ""
```

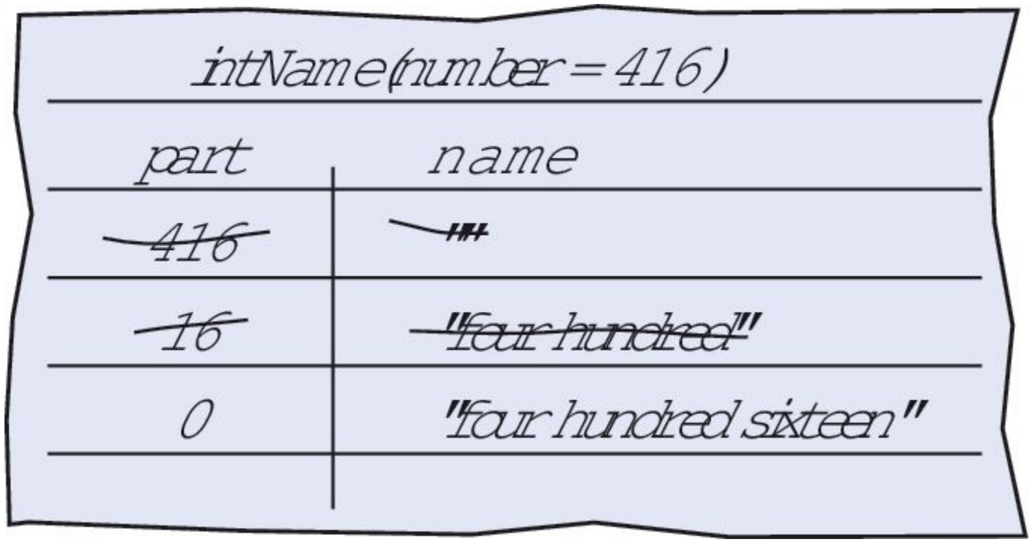
tensName



```
def tensName(number) :
    if number >= 90 : return "ninety"
    if number >= 80 : return "eighty"
    if number >= 70 : return "seventy"
    if number >= 60 : return "sixty"
    if number >= 50 : return "fifty"
    if number >= 40 : return "forty"
    if number >= 30 : return "thirty"
    if number >= 20 : return "twenty"
    return ""
```

Programming Tips

- Keep functions short
  - If more than one screen, break into ‘sub’ functions
- Trace your functions
  - One line for each step
  - Columns for key variables
- Use Stubs as you write larger programs
  - Unfinished functions that return a ‘dummy’ value



intName(number = 416)

part	name
~~416~~	~~""~~
~~16~~	~~"four hundred"~~
0	"four hundred sixteen"