Introduction to:
Computers & Programming
Defining Identifiers:
Objects with Names

Adam Meyers

New York University

Outline

- The types of objects with names
  - Functions, Variables, Programs, Modules, etc.
- Defining functions
- Defining variables
- The Scope of Variables
- Introducing For Loops
- Summary

Identifiers (Things with Names)

- Variables – pointers to objects (values of variables). When a variable is assigned a new value, it points to a new object.
- Functions
  - mappings from input to output
  - blocks of code that perform actions
  - or both
- Library Files (Python Modules)
  - Functions and variables in a (.py) file
  - Kept in special library folders
    - MAC OS – /Library/Python/3.1/site-packages/
    - Windows – c:\python31\lib\
- Programs (Python scripts) –
  - (.py) files like modules, but have 1 main function
  - /Applications/Python\ 3.1/Extras/Demo/turtle/tdemo_planet_and_moon.py

Limits on Function/Variable Names

- Cannot be identical to Python keywords
  - For a list of keywords type the following in IDLE

    import keyword

    keyword.kwlist
- Must begin with a letter (**a, b, A, B**) or underscore (_)
- Other characters can be letters, numbers or _
- Illegal names:
  - 5ways_to_do_it **(First character is not a letter)**
  - the\ big\ bad\ wolf **(Spaces and slashes are not allowed)**
  - import **(Python keywords cannot be redefined)**

Good Practices for Choosing Names

- Choose names that clearly represent the role of the function, variable, etc. in the code
- Use underscores _ instead of spaces in long names
- OR use the ***camelCase*** convention, in which uppercase letters signify the beginning of new words

An Example from Physics

- def get_distance_traveled (speed, time):

  print('Calculating the distance traveled given speed of ', speed, 'and time of ', time)

  return (speed * time)
- get_distance_traveled is a function
  - Models a mathematical function from the pair ***(speed, time)*** to the distance traveled if an object travels at that speed for that amount of time.
  - Prints what the function is modeling as a side effect
  - Parameters ***speed*** and ***time*** model speed (e.g., in km per hour) and time (e.g., in hours)

Dissection of get_distance_traveled

- Line 1: def get_distance_traveled (speed, time):
  - "def" begins a function definition
  - Next, is the name of the function (get_distance_traveled)
  - Then are zero or more variables
    - Surrounded by parentheses and separated by commas
  - Ends in a colon (which always precedes a block of statements)
  - The indent of the rest of the function indicates it forms a block
    - All the statements within a block are indented the same
- Line 2: a print statement (a side effect) – executes 1st
- Line 3: return(speed * time) – executes 2nd
  - return 'returns' its value from a block
    - In this case, that means it determines the value of the function

Why is get_distance_traveled easy to read and to use?

- The names of the function and the variables
  - Are based on the role that they play
  - Are spelled out and not abbreviated
- The print statement tells the user what is going on
- It is a very simple function
- Logically equivalent, but hard to understand
  - def gds (s, t):

    return (s* t)
  - Other people looking at the code would not understand what it is supposed to mean
  - The author would not understand it a month after writing it

Not all (Python) Functions have Input or Output

- This function has neither input nor output

```
def why_D_Cheney_should_be_in_jail():
    print('He used a firearm while intoxicated, a felony')
```

- This function has input, but no output

```
def print_something_3_times(that_thing):
    print(that_thing, that_thing, that_thing)
```

- This function has output, but no input

```
def the_secret_of_life()
    return(42)
```

Choosing a Good Function Name

- It should be legal:
  - Begin, with a letter or _
  - Contain only letter, numbers and _
  - Not match a Python Keyword
- The name should not already be used for a different function
  - You may unintentionally redefine a function or it just may be confusing
- The name should be self-explanatory
  - One reading the code should understand its purpose

Variables

- A variable points to another object (its value)
- The object pointed to changes if a variable is assigned a new value
- These objects typically include integers, floats, strings, among other types (the variable has the same type as its value)
- Variables have scope, which determines when they are active
- Parameters of a functions are variables whose scope is the function (they are only active while the function is active) (There are other local variables as well).

Some Properties of Variables in Other Languages

- Some languages require variable to have permanent types
  - This improves efficiency and may make debugging easier
  - Is a little difficult to get used to
- Some languages allow variables to point to functions
  - For example, in LISP it is possible to write a function that takes a function as one of its arguments and applies that function to another one of its arguments
- Some languages formally distinguish between constants and variables
  - Constants are unchangeable in such languages
  - In contrast, there is no way in Python to prevent a programmer from (for example), setting PI to a new value

Initializing/Setting Variables 1

- The following will produce an error in Python

  this_year

  – because ***this_year*** is not a legal Python object
- Now let's execute the following Python statement:

  this_year = 2010

  – This initializes this_year as a variable

  – This sets this_year to 2010
- Now there is no error when we type

  this_year

Initializing/Setting a Variable 2

- Python's “=” operator assigns a value to a variable
  - my_age = 47
  - my_name = 'Adam'
  - my_age
  - my_name
- The first such assignment initializes the variable
  - It assigns it a data type (try the function 'type')
  - It brings the variable into existence
  - Initialization is a separate step in some programming languages

Choose a Good Variable Name

- It is legal:
  - It begins with a letter or _ and contains only numbers, letters and _
  - It does not match a python keyword
- It does not share the same name as some other variable
  - More on next few slides
- Good variable names are self-explanatory
  - character_name = 'Mr. Gumby' ## good variable name
  - cn = 'Mr. Gumby' ## bad variable name
- Good variable names make it easy for anyone to understand your program
  - A teacher grading your program
  - A collaborator on a programming project
  - Yourself if you want to modify code you wrote 6 months ago
- Good variable names helps minimize the number of comments that are needed to make code readable

The scope of variables 1

- this_year from the last slide is a global variable
- To illustrate this, let's see how different functions handle the variable this_year
- The simplest case:

```
def print_this_year ():
    print(this_year)
```

  – Executing it prints 2010
- Notice that the variable assignment took place outside of any function, but was valid inside

The scope of variables 2

- Now for a less straight-forward case:

```
def mess_with_this_year ():
    this_year = 'the year one'
    print(this_year)
```

- After executing this function, what is the value of the variable ***this_year***?
  - Should it be 'the year one'?
  - Should it be 2010?
  - Why?

Variables: Global vs. Local

- A global variable definition holds everywhere
- By Default, variables assigned within functions are 'local' to those functions
- Figuring out how variables are interpreted requires that we know where they were defined
- In Python, local variable definitions take priority
  - By default, assignment statements inside functions do not effect global variables

Changing Global Variables Inside Functions

- To change global variables inside functions

```
def globally_mess_with_this_year ():
    global this_year
    this_year = 'the year one'
    print(this_year)
```

  – The keyword ***global*** lets one change global variables
- Now let's try this function
- Note that, in Python, changing global variables requires extra effort – it is not the normal case

Scope When Functions Call Functions

- Suppose we have the following functions

```
def praise (input_string):
    return(input_string+' is nice')
def disagree (input_string):
    return('I disagree with the statement: ' + input_string)
```

- Let's try using them separately and together

```
praise('coffee')
disagree('Fred is nice')
disagree(praise('coffee'))
```

What happened to the variable input_string?

- There are 2 separate variables with the same name
  - The parameter of the function ***praise***
  - The parameter of the function ***disagree***
- Each variable is local to its function
  - It only 'exists' while the function is 'active'
- When we called ***disagree(praise('coffee'))***
  - The function ***praise*** activated first and its variable ***input_string*** was bound to ***'coffee'***
  - Then, the function returned 'coffee is nice' and its variable (called ***input_string***) ceased to exist
  - Then the function ***disagree*** activated with its variable (called ***input_string***) bound to 'coffee is nice'
  - The function returned 'I disagree with the statement: coffee is nice' and the variable ***input_string*** that is local to ***disagree*** ceased to 'exist'.

In Python, functions pass variables by value, not by reference

- Let's set a global variable and a function

```
my_name = 'Adam'
def my_name_is (name):
    print('My name is ', name)
```

- Now let's call the function with the global variable as an argument:

```
my_name_is(my_name)
```

- The value of ***my_name*** ('Adam'), not the pointer, is the argument of ***my_name_is***

What Would it Mean to Pass by Reference?

- It would mean that you were passing the pointer rather than the value
- It would mean that it was possible to change the value of a global variable or even switch values
- For example, a function that trades values of global variables cannot work in Python (without using the keyword ***global***)

```
def switch_variable_values (variable1, variable2):
    place_holder = variable1
    variable1 = variable2
    variable2 = place_holder
```

- This is because variable1 and variable2 represent values, not variables
- NB: It would be possible to change global variables using the global keyword, but not variable bindings in general

One Other Kind of Parameter

- So far we have defined arguments of functions by the order in which they occur
  - function(arg1,arg2,....)
  - speed(distance,time)
- However, it is also possible to use keyword arguments after all other arguments
  - print(arg1,arg2,arg3,....,sep=' ',end='\n')
  - Keyword arguments have default values

Examples

- def print_multi_char(number,char='*'):

  print(char*number)
- By default character is an *, but you can set it to anything.
- def print_date(month,day,year=2011):

  print(month, str(day)+',', year)
- By default year is set to 2011

Introducing Loops: The For Loop

- for number in range(5):

  print(number)
- Characteristics of a for loop:
  - 1st line: ***for*** + variable name + ***in*** + sequence + ***:***
  - The rest of the loop: is called the body and is indented under the 1st line
  - The body of code repeats once for each item in the sequence
  - The loop variable (e.g., number) has the following properties:
    - The variable is reset upon each iteration of the loop
    - The variable is set to the next item in the sequence
    - The scope of the variable is the loop
- Loops are a type of "control" structure used to make pieces of code repeat
- This term, we will focus mainly on ***for*** loops (now) and ***while*** loops (later)

Examples of Loops

- Loop Example 1

```
total = 0
for number in range(1,6):
    total = total + number
```

  – What is the value of total?
- Loop Example 2

```
for letter in 'abcdefg':
    print(letter)
```

Loop example 3

- def put_stars_around_letters (word):

```
    output = '*'
    for letter in word:
        output = output+letter+'*'
    return(output)
```

- Variables are often used in conjunction with loops (output, total, etc.) to accumulate the result. These are called accumulator variables.

Summary 1

- Identifiers are the named objects used to write programs
  - The syntactic rules of the programming language determine what legal names are
    - Python variable and function names must not duplicate Python keywords, must begin with a letter or _, and can only contain certain characters (letters, numbers, and _)
  - Syntactic rules also determine how identifiers are initialized and defined
- Other conventions make one's code readable
  - Choosing meaningful names
  - Not reusing names too much
  - Similar concerns for filename selection, good jargon in science, etc.

Summary 2

- Predicting how variables are interpreted depends on understanding issues of variable scope
- Variables defined in a function are local to that function—they only 'exist' inside the function
  - Unless they are specifically tagged as being global
- Variables defined outside a function are global--they are always 'active' unless there is an active local variable with the same name
- Using different names for local and global variables may make code easier to understand

Summary 3

- Some special structures in python introduce blocks of text. Two examples:
  - function definitions
  - loops
- These structures include:
  - A statement introducing the structure ending in a colon (:)
  - The body indented below this line
- These structures sometimes introduce variables
  - By default, the structures are the scope of the variables they introduce

Homework: Due In Two Classes Page 1

- Read Chapter 3
- Create a .py file that includes all the functions describe below and on the next page.
- 1. Write a mathematically oriented function of your choice that calculates something relevant to math or science.
    - Define a global variable and set it to the output of call to this function.
    - Example: temp_today = faren_to_celcius(35)

Homework Page 2

- 2. Write a function:
  - Takes one or more strings as arguments
  - Alters the strings by using the concatenation operator '+' in some way.
  - Prints helpful messages and returns the altered string
  - Example: month_day_year('January',31,2011)
    - Causes the following 3 lines of print:
      - Month = January
      - Day = 31
      - Year = 2011
    - Then it returns the string 'January 31, 2011'

Page 3

- Question 3: Run the following code in IDLE:

```
output = 1
for number in range(1,5):
    output = output*number
```

  - Add the following to the bottom of your .py file that contains the functions for questions 1 and 2:
    - A comment labeled # 3.
    - It should explain why output has the value that it does.

How Homework Is Graded

1. Does the code run?
2. Does it fulfill the assignment, answer the question asked, etc.?
3. Is it clear?
   - Do you use good variable names
   - Do you use comments as appropriate
4. When given latitude, do you choose a really easy example, a medium example, a creative or interesting example, etc.? I.e., Bells and whistles are never required, but they may help your grade.