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
• 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\n
- 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
    
    ```python
    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)**
  ```python
  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 1\textsuperscript{st}
• Line 3: return(acceleration * time) – executes 2\textsuperscript{nd}
  – 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
  ```python
def why_D_Cheney_should_be_in_jail():
    print('He used a firearm while intoxicated, a felony')
  ```

- This function has input, but no output
  ```python
def print_something_3_times(that_thing):
    print(that_thing, that_thing, that_thing)
  ```

- This function has output, but no input
  ```python
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)
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:
  ```python
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:
  ```python
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

```python
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
  ```python
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
  ```python
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'.

Intro to: Computers & Programming
Defining Identifiers: Objects with Names
V22.0002-001
In Python, functions pass variables by value, not by reference

• Let's set a global variable and a function

```python
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:

```python
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
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, 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
Homework, Due Next Week

• Read Chapter 3
• Create a .py file as follows
  – Define global and local variables.
  – Name all identifiers so the code is understandable
  – Write 3 functions (or more)
    • Write a function that prints a string
    • Write a function that uses math, e.g.,
      – Compute a statistic, a physics formula, etc.
    • Write a function that combines strings together, e.g.,
      – Fills in blanks in sentences with variables