CMPUT 274

Functions

Topics Covered:

- Define your own function
- Local vs Global variables
- Default parameters
- docstring

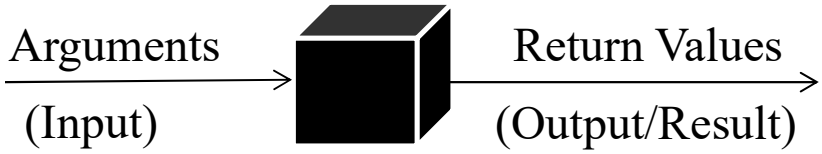
What is a Function?

- **Function**: a group of statements that performs a specific task
- Often used to break down programs into small, manageable pieces
  - Organizes code
  - Result: code is easier to read, test, and debug
- Benefits:
  - Can reuse chunks of code
  - Update in one place → less chance of introducing error
  - Easy way to split up work on a team

Built-in Functions

- Good news: we've already been using functions
  - Examples: `input()`, `print()`, `int()`, `len()`

- We don't need to know exactly how these functions are implemented to be able to use them.



- Remember: use help() to view documentation

- For full list of built-in Python functions: https://docs.python.org/3/library/functions.html

Define Your Own Function

Meaningful name

Optional: way to pass **input** to function

```
def function_name(parameters):
    statement1
    statement2
    :
    statementN
    return values
```

Notice **colon** at end of function *header*

Notice **parentheses** immediately after function name

All statements in the function *body* must be **indented one level**

Optional: way to pass **output** back to where function was called. **None** object is returned if value not specified.

Simple Example

```
print("Twinkle, twinkle, little star")
print("How I wonder what you are")
print("Up above the world so high")
print("Like a diamond in the sky")
print("Twinkle, twinkle, little star")
print("How I wonder what you are")
```

Repeating, non-adjacent lines

```
def displayChorus():
    print("Twinkle, twinkle, little star")
    print("How I wonder what you are")

# main script
displayChorus()
print("Up above the world so high")
print("Like a diamond in the sky")
displayChorus()
```

Pass Arguments, Return Value

```
# function to add/join 3 values together
def plus3(val1, val2, val3):
    print('In plus3: ', end='')
    print('val1={},'.format(val1), end='')
    print('val2={}, val3={}'.format(val2, val3))
    result = val1 + val2 + val3
    return result

# main script
total = plus3(12, 3, 5)
print('In main: total is', total)
word = plus3('a', 'b', 'c')
print('In main: concatenated string is', word)
```

Save returned value by assigning to a variable

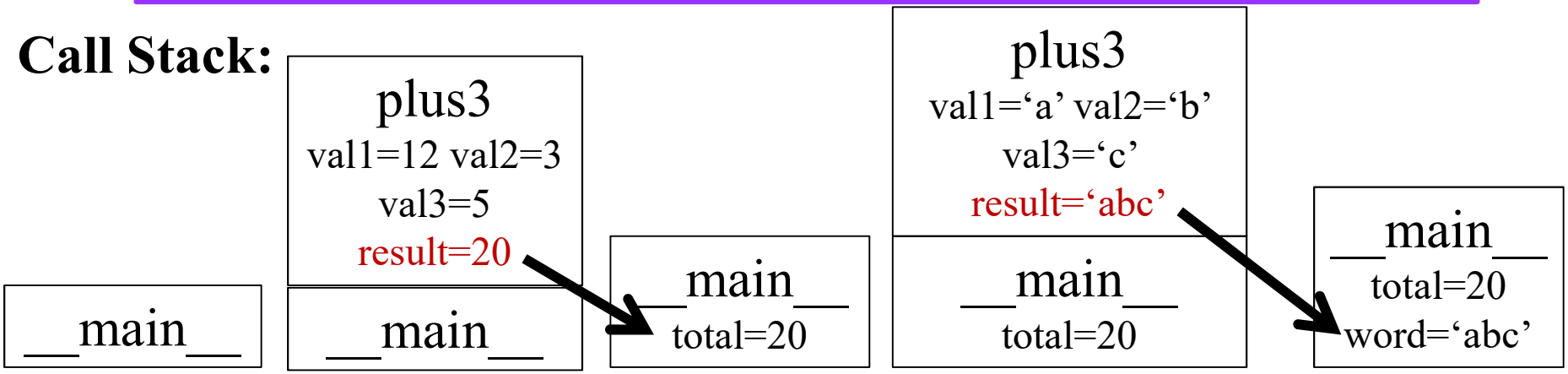
```
In plus3: val1=12,val2=3, val3=5
In main: total is 20
In plus3: val1=a,val2=b, val3=c
In main: concatenated string is abc
```

Calling a Function

```
# function to add/join 3 values together
def plus3(val1, val2, val3):
    print('In plus3: ', end='')
    print('val1={},'.format(val1), end='')
    print('val2={}, val3={}'.format(val2, val3))
    result = val1 + val2 + val3
    return result

# main script
total = plus3(12, 3, 5)
print('In main: total is', total)
word = plus3('a', 'b', 'c')
print('In main: concatenated string is', word)
```

Call Stack:



"__main__" Namespace

- "__main__" is name of scope where top-level code executes
- "__main__" namespace contains global variables
- When a Python program is run, the built-in variable __name__ is set to "__main__"
- Convention to include the following in your program:

```
if __name__ == "__main__":
    # code to run when this is main program
```

- Allows us to run main script, or to import functions from the file to use in another file's program

Local vs Global Variables

- A global variable is created in main script of a file
  - Accessible throughout file (and in any file which imports that file), including inside any function definitions
- A local variable is created inside a function
  - Not accessible outside the function
  - Automatically destroyed when function ends
- Different functions can use same names for their respective local variables
  - Different variables (exist in different namespaces)
  - Python checks for local variables first, then global
  - Can cause confusion!

Example: Local Variables

```
def canada():
    cows = 12.255 # million
    print('In Canada, there are', cows, 'million cows.')

def alberta():
    cows = 4.915 # million
    print('In Alberta, there are', cows, 'million cows.')

def ireland():
    print('In Ireland, there are', cows, 'million cows.')

# main script
if __name__ == "__main__":
    cows = 6.5935 # million
    canada()
    alberta()
    ireland()
```

canada() has **local** variable *cows*

alberta() has **local** variable *cows*

ireland() uses **global** variable *cows*

```
In Canada, there are 12.255 million cows.
In Alberta, there are 4.915 million cows.
In Ireland, there are 6.5935 million cows.
```

Updating Global Variables

```
def ireland():
    cows += 0.41
    print('In Ireland, there are', cows, 'million cows.')

# main script
if __name__ == "__main__":
    cows = 6.5935 # million
    ireland()
```

ireland() tries to use **global** variable *cows,* but ends up trying to create **local** variable cows when updating immutable object

```
UnboundLocalError: local variable 'cows'
referenced before assignment
```

```
def ireland():
    global cows
    cows += 0.41
    print('In Ireland, there are', cows, 'million cows.')

# main script
if __name__ == "__main__":
    cows = 6.5935 # million
    ireland()
```

Force ireland() to use **global** variable *cows*

```
In Ireland, there are 7.0035 million cows.
```

Updating Global Variables

- **BEWARE:** Since ANY function can change the value of a global variable, it's hard to keep track of just what it's storing

- Avoid using a global variable when a local variable makes more sense
  - Pass arguments to functions
  - Create local variables with unique names

Parameters: Default Values

- A parameter can be assigned a default value in function definition
  - e.g. `def function_name(param1 = value1):`
  - If corresponding argument is not provided when function is called, default value will be used

- Once one parameter is given a default value, all parameters to the right must have default values
  - e.g. `def function_name(p1, p2=val2, p3=val3):`
  - ERROR: `def function_name(p1=val1, p2, p3):`

Example: Default Values

```
def greet(name, salutation='Hello'):
    print(salutation + ', ' + name)

if __name__ == "__main__":
    greet('Alice', 'Good morning')

    # use default value for salutation
    greet('Bob')

    # key word arguments, out of order
    greet(salutation='Hi', name='Charlie')
```

```
Good morning, Alice
Hello, Bob
Hi, Charlie
```

docstring

- Good practice to include comment header for each function
  - Describe what function does, its arguments, what it returns, etc.
  - See Code Submission and Style Guidelines on eClass
- Format as multi-line docstring, using " " "
- Comment must start on line just below function header
- Contents of docstring are displayed when help(*function_name*) is called