Functions

Contents

1. Getting started with functions
2. Going further with functions

Demo folder: 04-Functions

1. Getting Started with Functions

- Simple functions
- Passing arguments to a function
- Returning a value from a function
- Understanding scope

Simple Functions (1 of 2)

- A function is a named block of code
  - Starts with the `def` keyword…
  - Followed by the name of the function…
  - Followed by parentheses, where you can define arguments…
  - Followed by a block, where you define the function body

```
def name_of_function(arg1, arg2, …, argn) :
  statements
  statements
  …
```

- To call a function
  - Specify the function name…
  - Followed by parentheses, where you can pass arguments

```
name_of_function(argvalue1, argvalue2, …, argvaluen)
```

Simple Functions (2 of 2)

- Here's an example of how to define and call simple functions

```
def say_goodmorning():
  print("Start of say_goodmorning")
  print(" Good morning!")
  print("End of say_goodmorning\n")

def say_goodafternoon():
  print("Start of say_goodafternoon")
  print("  Good afternoon!")
  print("End of say_goodafternoon\n")

def say_goodevening():
  pass


# Usage (i.e. client code)
say_goodmorning()
say_goodafternoon()
say_goodevening()

f = say_goodmorning
f()                    # Calls say_goodmorning() really

print("THE END")
```

simplefunctions.py

Passing Arguments to a Function

- You can pass arguments to a function
  - In the function definition, declare the argument names in the parentheses
  - In the client code, pass argument values in the call

- Example

```
def display_message(message, count):
  for i in range(count):
    print(message)

# Usage (i.e. client code)
display_message("Hello", 3)
display_message("Goodbye", 1)
```

functionarguments.py

Returning a Value From a Function

- Functions can return a value, via a `return` statement
  - If you don't return a value explicitly, the function returns `None`
- Example:

```
def display_message(msg):
  print(msg)

def generate_hyperlink(href, text):
  return "<a href='{0}'>{1}</a>".format(href, text)

def get_number_in_range(msg, lower, upper):
  while True:
    num = int(input(msg))
    if num >= lower and num < upper:
      return num


# Usage (i.e. client code)
result1 = display_message("Hello world")
print("result1 is %s" % result1)

result2 = generate_hyperlink("http://www.bbc.co.uk", "BBC")
print("result2 is %s" % result2)

result3 = get_number_in_range("Favourite month? ", 1, 13)
print("result3 is %s" % result3)
```

functionreturn.py

Understanding Scope (1 of 2)

- If you declare a variable outside a function:
  - The variable is global to the module
  - Prefix the name with ___ to make it private to this module
- If you declare a variable inside a function:
  - The variable is local to the function
- If you want to assign a global variable inside a function:
  - You must declare the variable inside the function, using the `global` keyword
  - Tells the Python interpreter it's an existing global name, not a new local name

Understanding Scope (2 of 2)

- This example shows how to define and use global variables

```
__DBNAME = None

def initDB(name):
  global __DBNAME
  if __DBNAME is None:
    __DBNAME = name
  else:
    raise RuntimeError("Database name has already been set.")

def queryDB():
  print("TODO, add code to query %s" % __DBNAME)

def updateDB():
  print("TODO, add code to update %s" % __DBNAME)


# Usage (i.e. client code)
initDB("Server=.;Database=Northwind")
queryDB()
updateDB()
```

globals.py

2. Going Further with Functions

- Default argument values
- Variadic functions
- Passing keyword arguments
- Variadic keyword arguments
- Built-in functions
- Examples of using functions

Defining Default Argument Values

- You can define default argument values for a function
  - In the function definition, specify default values as appropriate
  - In the client code, pass argument values or rely on defaults

- Example:

```
def book_flight(fromairport, toairport, numadults=1, numchildren=0):
  print("\nFlight booked from %s to %s" % (fromairport, toairport))
  print("Number of adults: %d" % numadults)
  print("Number of children: %d" % numchildren)

# Usage (i.e. client code)
book_flight("BRS", "VER", 2, 2)
book_flight("LHR", "VIE", 4)
book_flight("LHR", "OSL")
```

functiondefaultarguments.py

Variadic Functions

- Python allows you to define a function that can take any number of arguments
  - In the function definition, prefix the last argument name with *
  - Internally, these arguments will be wrapped up as a tuple
  - You can iterate through the tuple items by using a `for` loop

- Example

```
def display_favourite_things(name, *things):
  print("Favourite things for %s" % name)
  for item in things:
    print("  %s" % item)

# Usage (i.e. client code)
display_favourite_things("Andy", "Jayne", "Emily", "Thomas", 3, "Swans", "Skiing")
```

functionvariadicarguments.py

Passing Keyword Arguments

- Client code can pass arguments by name
  - Use the syntax argument_name = value
- Useful if the function has a lot of default argument values
  - Client code can choose exactly which arguments to pass in
- Example:

```
def book_flight(fromairport, toairport, numadults=1, numchildren=0):
  print("\nFlight booked from %s to %s" % (fromairport, toairport))
  print("Number of adults: %d" % numadults)
  print("Number of children: %d" % numchildren)

# Usage (i.e. client code)
book_flight("BRS", "VER", 2, 2)
book_flight("LHR", "CDG", numchildren=2)
book_flight(numchildren=3, fromairport="LGW", toairport="NCE")
```

functionkeywordarguments.py

Variadic Keyword Functions

- It's also possible to define variadic keyword arguments
  - Use ** rather than * on the argument
  - Allows you to pass in any number of keyword args

- Internally, the arguments are wrapped as a dictionary
  - You can iterate through the key/value pairs by using a `for` loop

- Example

```
def myfunc(**kwargs):
  for k, v in kwargs.items():
    print ("key %s, value %s" % (k, v))

# Usage (i.e. client code)
myfunc(favTeam="Swans", favNum=3, favColour="red")
```

functionvariadickeywordarguments.py

Built-In Functions

- Python has a suite of built-in functions that are always available

Built-in Functions				
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Examples of Using Functions (1 of 2)

- We've written some examples to illustrate how to use functions in realistic scenarios
  - Processing lines of text from a file
  - Using regular expressions to find particular values in the file

- Demo location
  - `C:\PythonDev\Demos\04-Functions\WorkedExamples`

Examples of Using Functions (2 of 2)

- To open and read a file:
  - Call `open()` to open a file - returns a file handle
  - To read lines from the file, simply iterate over the file handle

- To use regular expressions:
  - The `re` module has `compile()` and `search()` functions to compile and use a regular expression

- Here's the first example:

```
import re

pattern = re.compile('Attribute ID \(0xC2\)')

with open('data.txt') as fh:
    for line in fh:
        result = pattern.search(line)
        if result:
            print(line)
```

read_data1.py

Any Questions?