SNS College of Engineering
Department of Information Technology

Comments/Modules/ Functions

Presented By
M.Malarmathi
AP/IT

Comments

- Program is bigger, more complicated and difficult to read , figure out what it is doing
- To over come this, it is good idea to add notes to your programs to explain the flow of execution
- These notes are called **comments, and they start with** # symbol

Example

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
percentage = (minute * 100) / 60 # percentage of an hour
```

- The line starts with # has no effect on the program
- Comments are most useful when they document non-obvious features of the code
- The reader can figure out what the code does
- it is much more useful to explain why.
  - v = 5 # assign 5 to v
  - v = 5 # velocity in meters/second.
- Good variable names can reduce the need for comments
- long names can make complex expressions hard to read.

Modules

- Modules are simple way to structure the program
- In python Modules are used by importing them
  - import math
  - print math.sqrt(10)

```
from string import whitespace
from math import *
from math import sin as SIN
from math import cos as COS
```

Functions

- A **function is named as a sequence of statements that performs** a computation
  - >>> type(32)
  - <type 'int'>
- The name of the function is type. The expression in parentheses is called the **argument of** the function
- To define a function, specify the name and the sequence of statements
- Later, “call” the function by name
  - def print_lyrics():
  - print("I'm a lumberjack, and I'm okay.")
  - print("I sleep all night and I work all day.")

- def is a keyword that indicates that, this is a function definition
- The name of the function is print_lyrics.
- The rules for function names are the same as for variable names
- letters, numbers and underscore are legal, but the first character can't be a number
- Avoid using variable and function in the same name
- The empty parentheses after the name indicate that this function doesn't take any arguments.

- The first line of the function definition is called the **header,** the rest is called the **body**
- The header has to end with a colon and the body has to be indented
- Indentation is always four spaces. The body can contain any number of statements
- The strings in the print statements are enclosed in double quotes. Single quotes and double quotes do the same thing

```
>>> def print_lyrics():
... print "I'm a lumberjack, and I'm okay."
... print "I sleep all night and I work all day."
...
```

- The syntax for calling the new function is the same as for built-in functions:

```
>>> print_lyrics()
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

- example, to repeat the previous refrain, write a function called repeat_lyrics:

```
def repeat_lyrics():
print_lyrics()
print_lyrics()
```

- And then call repeat_lyrics:
- >>> repeat_lyrics()

```
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
I'm a lumberjack, and I'm okay.
I sleep all night and I work all day.
```

```
def print_lyrics():
print "I'm a lumberjack, and I'm okay."
print "I sleep all night and I work all day.“

def repeat_lyrics():
print_lyrics()
print_lyrics()
```

- program contains two function definitions: print_lyrics and repeat_lyrics
- Function definitions get executed just like other statements, but the effect is to create function objects.
- The statements inside the function do not get executed until the function is called, and the function definition generates no output.

Flow of Execution

- The order in which statements are executed, is called the **flow of execution**
- Execution always begins at the first statement of the program.
- Statements are executed one at a time(Top down approach)
- Function definitions do not alter the flow of execution, the function are not executed until the function is called

- If the function is called, the flow jumps to the body of the function, executes all the statements .
- then comes back to pick up where it left off

Parameters and arguments

- Inside the function, the arguments are assigned to variables called **parameters.**
- when you call math.sin you pass a number as an argument
- Some functions take more than one argument: math.pow takes two, the base and the exponent
- Pow (base,exp)

- example for user-defined function that takes an argument

```
def print_twice(bruce):
print bruce
print bruce
```

- This function assigns the argument to a parameter named bruce
- When the function is called, it prints the value of the parameter twice

```
>>> print_twice('Spam')
Spam
Spam
>>> print_twice(17)
17
17
>>> print_twice(math.pi)
3.14159265359
3.14159265359
```

```
print_twice('Spam '*4)
Spam Spam Spam Spam
Spam Spam Spam Spam
>>> print_twice(math.cos(math.pi))
-1.0
-1.0
```

- You can also use a variable as an argument:

```
>>> michael = 'Eric, the half a bee.'
>>> print_twice(michael)
Eric, the half a bee.
Eric, the half a bee.
```

- When you create a variable inside a function, it is **local, which means that it only exists** inside the function. For example:

```
def cat_twice(part1, part2):
cat = part1 + part2
print_twice(cat)
```

- This function takes two arguments, concatenates them, and prints the result twice. Here is an example that uses it:

```
>>> line1 = 'Bing tiddle '
>>> line2 = 'tiddle bang.'
>>> cat_twice(line1, line2)
    Bing tiddle tiddle bang.
    Bing tiddle tiddle bang.
```

- When cat_twice terminates, the variable cat is destroyed. If we try to print it, we get an exception:

```
>>> print cat
NameError: name 'cat' is not defined
```

3.10.