Functions

Idea: Helps us build bigger programs by collecting code into re-useable units

Real Life Examples:

Capsule Coffee Maker - place a capsule and water (input) and get coffee (output)

Vending machine - place money and set choice (input) and get a treat (output)

Functions should perform a clearly defined, specific task

“Define once and use forever!”

Functions

Good functions act like a **blackbox** - the user doesn't need to know how the function works to use it

Functions are **abstractions**: they abstract the details so we can focus on the big picture

Functions allow us to write **modular** code. Modular code is organized in clearly defined sub-components. Each sub-component can be designed, implemented and tested independently

> Analogies: A car consists of independent modules such as lights, steering column, and brakes. A book consists of modules which build up such as sentences, paragraphs, sections, and chapters.

Function syntax

Syntax:

```
def <name>(<param1>,<param2>,...,<paramN>):

    <body>

    return <value>
```

colon

parameters, or arguments, or inputs. You can have any number of these, including none!

returning a value is optional

indent important

Aside - Terminology

Programmers use the terms **void**, **None**, and **NULL** to indicate nothing

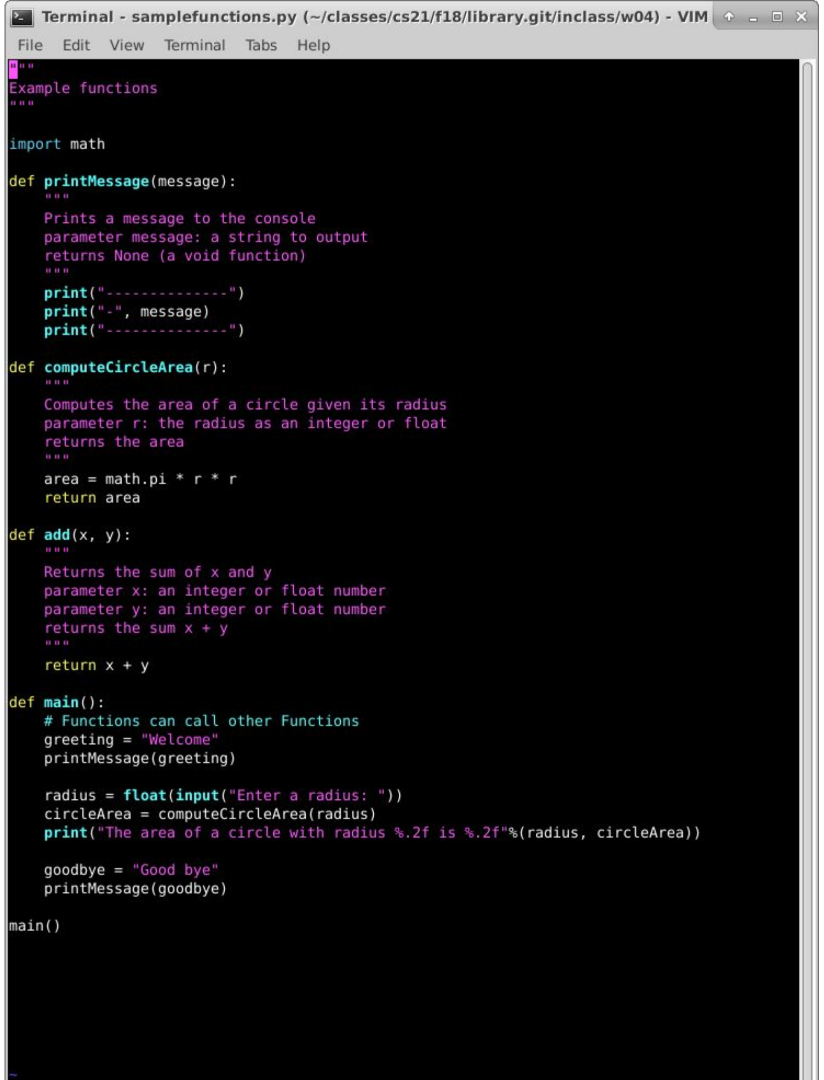
Ex: a function with no return value is sometimes called a **void function**

Ex. Python3 defines a special datatype called NoneType to represent variables that have nothing inside them

Function examples

You’ve been using functions already: min(), Math.sqrt(), main(), len(), input()

But you can also define your own!


Terminal - samplefunctions.py (~/classes/cs21/f18/library.git/inclass/w04) - VIM
File
Edit
View
Terminal
Tabs
Help

```python
"""
Example functions
"""

import math

def printMessage(message):
    """
    Prints a message to the console
    parameter message: a string to output
    returns None (a void function)
    """
    print("-------------")
    print("-", message)
    print("-------------")

def computeCircleArea(r):
    """
    Computes the area of a circle given its radius
    parameter r: the radius as an integer or float
    returns the area
    """
    area = math.pi * r * r
    return area

def add(x, y):
    """
    Returns the sum of x and y
    parameter x: an integer or float number
    parameter y: an integer or float number
    returns the sum x + y
    """
    return x + y

def main():
    # Functions can call other Functions
    greeting = "Welcome"
    printMessage(greeting)

    radius = float(input("Enter a radius: "))
    circleArea = computeCircleArea(radius)
    print("The area of a circle with radius %.2f is %.2f"%(radius, circleArea))

    goodbye = "Good bye"
    printMessage(goodbye)

main()
```

Advantages of functions

1. Re-useability - “define once and use forever”
2. Modularity - “top-down design”
   a. Split big problems into small, easy-to-solve problems
3. Easier to debug and maintain
   a. Cut & paste => bugs have to be fixed everywhere. Code in a function only has to be fixed once
4. Abstraction = “black box”
   a. Users do not need to know how it works in order to use it