Functions: Python

Sarah Khoja

FCM 708

Necessity of Functions

- With smaller snippets of code, the method we have so far is okay
  - Writing all the code within one file, and having it run line by line.
- Problem:
  - With larger pieces of code, this approach becomes terrible
    - Hard to read
    - Hard to understand
    - Hard to debug
    - Hard to make the pieces work well together

Functions:

- Goal of programming- reduce repetition to a minimum, by introducing functionality
- Functions- a way to achieve decomposition and abstraction

Functions and Abstraction

- Example- car engine
  - You don’t need to know how an engine works
  - Know the input/output
    - Fill car with gas, and the engine converts the gas into power
  - Idea of abstraction-
    - Don’t need to know how engine works to run the car

Functions and Decomposition:

- Car has many parts, the engine being one part
- The many parts each have their own task
  - tires
  - Steering wheel
  - Pedals
- But each part works with all the other parts **together** in order to make the car run.
- The concept of decomposition:
  - Different devices work together to achieve the same goal

Abstraction and Coding

- In order to explain abstraction in terms of programming:
  - Recall, you don't need to know how the engine works in order to use it
  - Similarly in programming,
    - I don't need to know how the function returns me an answer
    - I just give the correct input, and an output is produced
  - Achieve abstraction by leaving docstrings, which tell how to use the function, so programmers can simply use rather than figure out how it works.

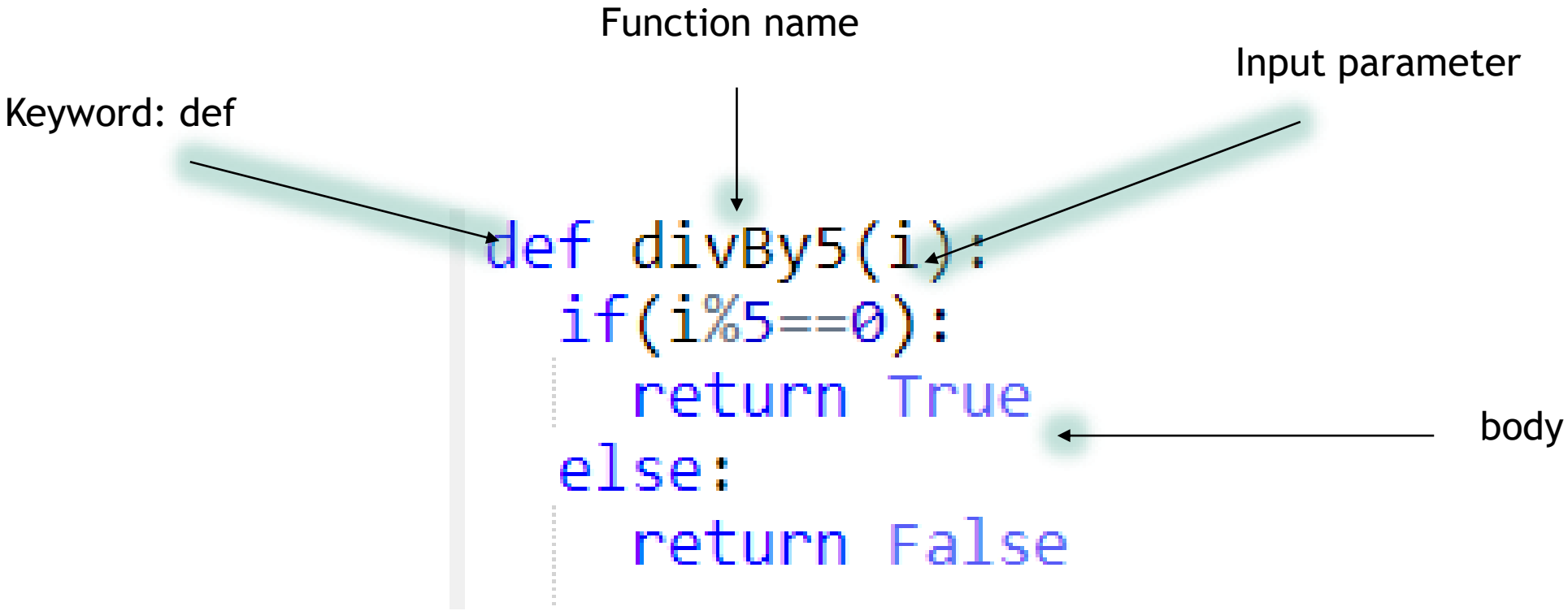
Decomposition and Coding

- Achieve structure in code by: Decoomposition
  - Breaking code into modules, which are:
    - Self contained
    - Used to break up code
    - Intended for reuse
    - Keeps organization
    - Coherent code
  - Make self contained code, which all have different functionality but work together to make the program complete.

Functions:

- Snippets of code that are reusable
- The functions are not used until called/invoked in the main program
- Functions:
  - Have a name
    - How to call it
  - Parameters (0 or more)
    - input
  - Has a body
    - What it does
  - Returns something
    - output

Function example



divBy5

```
def divBy5(i):
    if(i%5==0):
        return True
    else:
        return False
```

- What does this function do?
  - Takes an input i
    - Because i is being %5, then we assume (as does the python interpreter) that i is and integer
    - We check if i mod 5 is equal to 0
      - Meaning that i is evenly divisible by 5
      - If so, return true
    - If not
      - Return false
    - This function checks where an integer input, i, is divisible by 5.

Variable scope

```
def divBy5(i):
    if(i%5==0):
        return True
    else:
        return False
```

- Input value i:
  - This is the formal parameter
- When we later call this function in our program
  - Ex- divBy5(10) or divBy5(someInt)
  - someInt is the:
    - Actual parameter
- The actual parameter gets bound to the formal parameter when the function is called.
- Box anaology

Functions:

- You can also save the returned value
- When a function is called, after it executes, the last thing it does is return a value to the program that called it
- This value can be saved
- Example:
  - x=divBy5(someInt)
  - The returned value (True or False) is saved in x

Functions return:

- All functions return!
  - If you do not manually define a return stmt in your function, then Python automatically returns a special type:
    - None
      - Which means: the absence of a value

Functions and Parameters:

- Types of params:
  - No params
  - String, int, double, float, bool, etc.
  - Functions as parameters

```
def fn_a():
  print("Inside fn_a")
def fn_b(y):
  print("Inside fn_b")
  return y
def fn_c(z):
  print("Inside fn_c")
  return z()


print(fn_a())
print(5+fn_b(2))
print(fn_c(fn_a))
```

Functions as parameters:

```
def fn_a():
  print("Inside fn_a")
def fn_b(y):
  print("Inside fn_b")
  return y
def fn_c(z):
  print("Inside fn_c")
  return z()


print(fn_a())
print(5+fn_b(2))
print(fn_c(fn_a))
```

```
Inside fn_a
None
Inside fn_b
7
Inside fn_c
Inside fn_a
None
```

Scope:

- Scope defines the accessibility of variables, relative to your position in the code
- www.Pythontutor.com