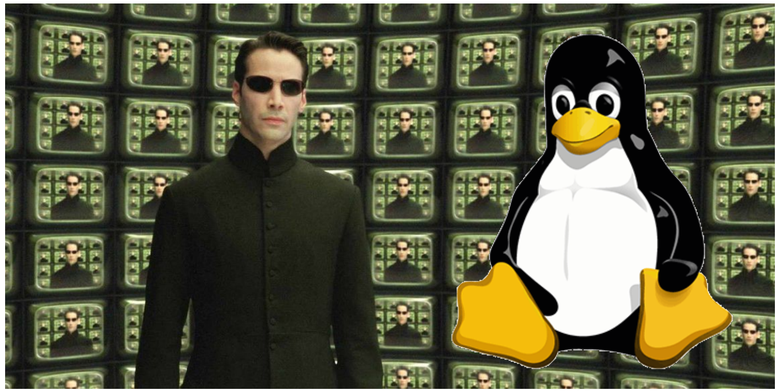
28.09.21 21:45

CSCI 120
Introduction to Computer Science and Programming I
Lecture 3: Functions I



Jetic Gū

Overview

- Focus: Basic Python Syntax
- Core Ideas:
    1. Function Declarations
    2. Python Scopes

Python Functions

Subroutine that you can call, anytime!

Existing Functions

- `input(...)`
  - Parameters (optional): `prompt`, a `str` to be printed before taking in input
  - Return: `str`, a single line of string from `stdin`
    - Return value: the result/output of the function, that can be used as part of an expression, or assigned to a variable
      e.g. `x = input()`

Review

Existing Functions

- `print(...)`
  - Parameters: objects, separated by ','
    - optional parameters: `sep`, `end`, ...

Review

`None` value

- `None` value
  - Check if a variable's value is `None`

```
if x is None:
    # do stuff...
```

  - Why do we need `None`?
    - For functions with no return value, such as `print(...)`

Concept

Python Function

- Functions are subroutines that can be reused
- Functions can be declared anywhere
- Functions can have parameters
- Functions can have return values

Concept

Functions Declarations

```
def welcome():
    print("Welcome to my programme")
```

Subroutine

```
welcome()
```

- Functions are declared using `def`
  - followed by function name (e.g. `welcome`)

    STYLE Function names should be lowercases, sometimes with underscores
  - parenthesis for parameters (in this case no parameter)
  - and colon, then a subroutine
- Functions can be called anywhere after it's declared

Technical

Functions Declarations

```
def welcome(firstname, lastname):
    print(firstname, lastname, ", I welcome you")

welcome("Jetic", "Gu")
# firstname gets "Jetic", lastname gets "Gu"
```

- Functions can have **parameters** (also called **arguments**)
  - Parameters are like new variables, you give these variables values when you call the function
  - Parameter variables only exist in the scope of the function's subroutine
  - You can have 0, 1 or any number of parameters

Functions Declarations

```
def sayHi(name):
    print("Hello,", name)
```

Function Declaration

```
def sayBye(name):
    print("Bye,", name)
```

Function Declaration

```
sayHi("Jetic")
sayHi("Jeremy")
sayBye("Jetic")
```

Main Programme

STYLE Always leave 2 empty line between function declarations, and your main programme

STYLE Always declare functions at the beginning of your `*.py` script files (before main programme)

Return a value

```
def sum_to(n):
    sum = 0
    for i in range(n + 1):
        sum += i
    return sum


print("The sum of 1 to 100 is", sum_to(100))
print("The sum of 1 to 1000 is", sum_to(1000))
```

- To return a value, or terminate the subroutine prematurely, use `return`

- When the programme reaches the function call, it will execute the subroutine, then get the value

Return a value

```
def sum_to(n):
    if n < 0:
        return
    sum = 0
    for i in range(n + 1):
        sum += i
    return sum
```

- To return a value, or terminate the subroutine prematurely, use `return`
  - When you just write `return`, it will exit the subroutine and return `None`
  - When your function exits without `return`, it will return `None`

Python Scopes

When and where can you access variables/functions

Python Scopes

```
stu_num = 0


def add_stu(name):
    print("Welcome", name)
    stu_num += 1

def del_stu(name):
    print("Bye", name)
    stu_num -= 1

add_stu("Jetic")
add_stu("Jack")
del_stu("Jetic")
print(stu_num)
```

- Where can you access variable `name`?
- Is `name` in `add_stu` and `del_stu` the same variables?
- Where can you access variable `stu_num`?

Python Scopes

Global Scope

```
stu_num = 0


def add_stu(name):
    print("Welcome", name)
    stu_num += 1

def del_stu(name):
    print("Bye", name)
    stu_num -= 1

add_stu("Jetic")
add_stu("Jack")
del_stu("Jetic")
print(stu_num)
```

Local Scope

Local Scope

- Variables (and functions) can only be accessed in their respective scopes, or their children scopes
- In this example, you have 3 scopes
- global scope (your script `main.py`)
  - local scope `del_stu`
  - local scope `add_stu`

Concept

Python Scopes

Global Scope

```
stu_num = 0


def add_stu(name):
    print("Welcome", name)
    stu_num += 1

def del_stu(name):
    print("Bye", name)
    stu_num -= 1

add_stu("Jetic")
add_stu("Jack")
del_stu("Jetic")
print(stu_num)
```

Local Scope

Local Scope

- A scope is created NOT when you write the code, but when you execute it
  - During the execution of this scope, new variables created are of the scope
- When a scope is deleted, all internal variables are lost

Concept

Python Scopes

Global Scope

```
stu_num = 0


def add_stu(name):
    print("Welcome", name)
    stu_num += 1

def del_stu(name):
    print("Bye", name)
    stu_num -= 1

add_stu("Jetic")
add_stu("Jack")
del_stu("Jetic")
print(stu_num)
```

Local Scope

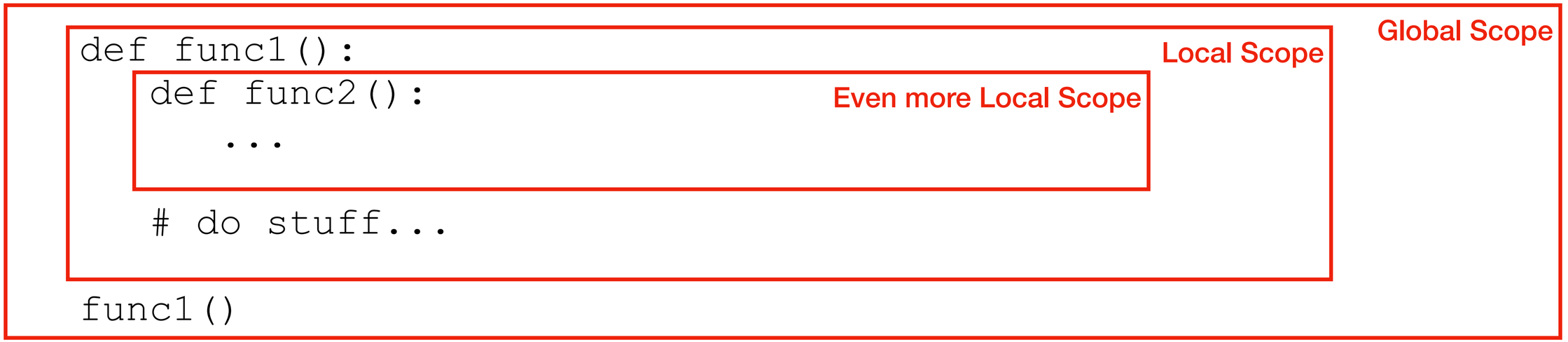
Local Scope

- You run python `main.py`
  The Global Scope is created
- You run `add_stu`
  Local scope `add_stu` is created
- `add_stu` finishes and returns
  Local scope `add_stu` is deleted
- You run `add_stu`
  Local scope `add_stu` is created
- `add_stu` finishes and returns
  Local scope `add_stu` is deleted

Different!

Technical

Python Scopes



- Python scope is different from other programming languages
  - A new subroutine doesn't create a new scope (unlike C, C++, etc.)
  - Functions declarations also have scopes
    - And yes, you can declare new functions inside a function, but that would limit the said function in a local scope

Concept

Python Scopes

Global Scope

```
def func1():
    def func2():
        ...

    # do stuff...

func1()
```

Local Scope

Even more Local Scope

- Calling `func1` inside `func1` is called a recursive function call
- You can call `func1` inside: main programme, `func1`, and `func2`
- You can call `func2` inside: `func1`, and `func2`
- You can NOT call `func2` outside of `func1`

Python Scopes

Global Scope

```
def func1():
    def func2():
        ...

    # do stuff...

func1()
```

Local Scope

Even more Local Scope

- Variable declared in the main script can be accessed by: main programme, `func1`, and `func2`
- Variables declared in `func1` can be accessed by: `func1`, and `func2`
- Variables declared in `func2` can only be accessed by: `func2`