CS65: Introduction to Computer Science

Functions
User defined functions

Md Alimoor Reza
Assistant Professor of Computer Science

Recap

- Variable syntax, naming convention
- Expression for complex calculation in a Python line
- Comments are helpful notes
- Getting inputs from user is accomplished by input()

Recap: Variable and assignment operator

- Variable is a named storage space in computer memory
- Need to use assignment operator (**=**) to store a value
- Location of assignment on the left
- Single value or some calculated value on the right
- **variable_name** = value

```
time_sec = 60
temp_degree = 27

mile_to_kilometer = 1.609
price_in_dollars = 1500.89
```

Numbers

```
first_name = "Md Alimoor"
last_name = "Reza"
```

Textual data

Expression

- A fragment of python code that calculates a new value called an expression

- For example, you can convert miles into meters using the following expression:

```
num_of_miles = 10
miles_to_kilometer = 1.609
```

```
num_of_meter = num_of_miles*miles_to_kilometer*1000
```

Exercise

- Can you compute the area of a rectangle?
  - Length of the two sides will be given in variables

- Can you compute the area of a circle?
  - Radius of the circle is given
  - Value of Pi is 3.14159

Getting Input from Users

- Built-in function in Python ***input("….")***
  - Step 1: displays the prompt to the user
  - Step 2: waits for user to type in something
  - Step 3: returns the typed content when user hits enter
  - Step 4: this value is stored if assigned to a variable

```
rect_a = input("enter the length of rectangle side a: ")
print(rect_a)
```

Demo

Errors (will be discussed more later)

- <u>Syntax error</u>
  - violating a programming language's rules on how symbols can be combined to create a program

- <u>Runtime error</u>
  - wherein a program's syntax is correct but the program attempts an impossible operation
    - dividing by zero
    - entering a string instead of an integer

Topics

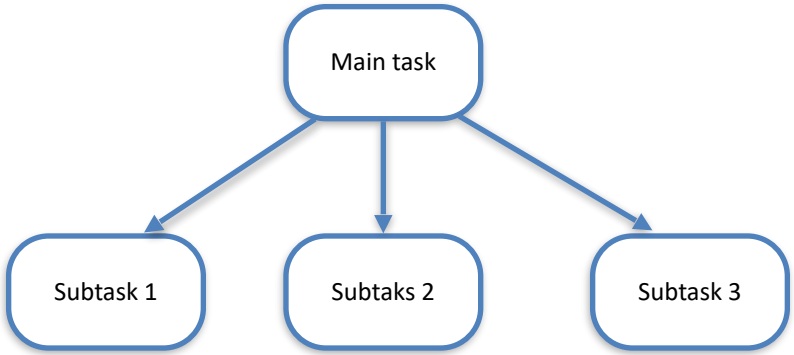
- Functions — a new concept
  - User defined functions vs built-in functions

- User defined functions
  - defining function: what statements it will execute
  - calling function: invoke/execute the defined body

Functions

- **Function** is a sequence of statements that performs a specific task — also called a subroutine

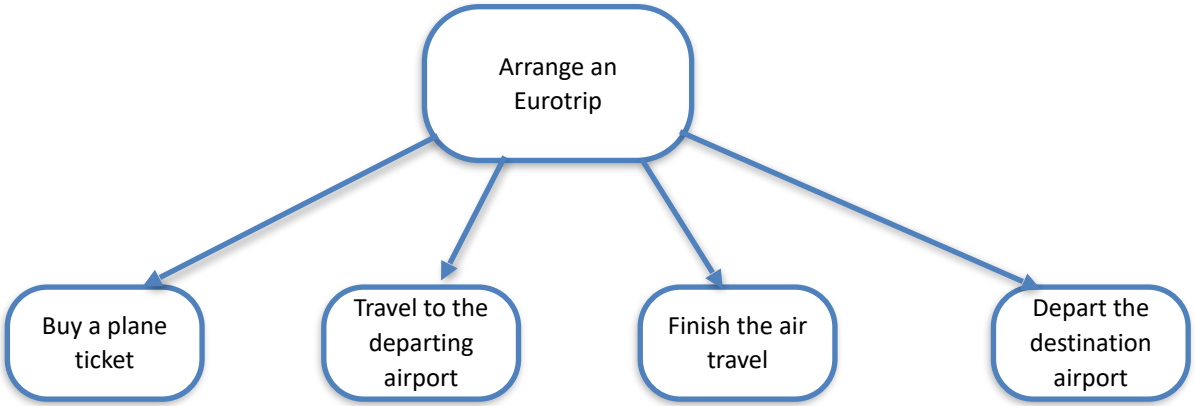
- Decompose a bigger task with the help of several smaller subtasks



Functions

- **Function** is a sequence of statements that performs a specific task — also called a subroutine

- Decompose a bigger task with the help of several smaller subtasks



You can write a python function for individual subtask!

Why should you use Functions?

- Decompose a bigger task with the help of several smaller subtasks
  - Code becomes more <u>modular</u> and <u>manageable</u>
    - Imagine, you need to write the same calculations over and over again *eg, 100 times*!
  - Code for a subtask can be <u>reusable</u>
  - <u>Individual member</u> in a team can write different functions
  - Improves <u>code readability</u>

Topics

- Functions — a new concept
  - User defined functions vs builtin functions

- A user defined function
  - <u>define</u>: what statements it will execute
  - <u>call</u>: invoke/execute the function body you <u>defined</u>

Functions

- **Function** is a sequence of statements that performs a specific task

- **Define** a function once
  - formula or template to solve a task with a series of statements
  - definition **doesn't do** anything unless it is called

- **Call** a function as many times as you like and receive return values
  - supply a matching signature to invoke an already defined function

Define a Function with no Parameters

```
def name_of_the_function() :

    statement 1
    statement 2
    …
    statement 100
    return expression
```

This line is called function header

- **name_of_the_function**: a meaningful name denoting the task with a preceding **def** keyword
- **statements**: a sequence of python instructions to be executed followed by an optional **return** keyword with expression(s)
    - without a **return** statement function implicitly returns **None**
- Notice: indention (eg, tab) is required to define a **function** and also notice at the end of the condition expression there is a **colon**

Define a Function with Parameters

```
def name_of_the_function(param1, param2, …, param4) :

    statement 1
    statement 2
    …
    statement 100
    return expression
```

Parameters

- Add a number of **parameters** as required for your task:
  - Parameters are <u>variables</u> used to exchange values during function call
  - <u>Values are mapped to parameters</u> each time the function is called
  - Parameters are <u>not available outside</u> the function

Demo: user defined function example

```
# Author's name: Md Alimoor Reza
# Author's contact: md.reza@drake.edu
# Date: (September 7, 2021)
# Collaborator:
#       self


# this user defined function adds two numbers
def add_numbers(num1, num2):
    sum = num1 + num2
    return sum
```

Shell

Watch out for these items

```
>>> a = 1
>>> b = 2
>>> res = add_numbers(a, b)
>>> print("sum of", a, " and", b, ":",res)

sum of 1  and 2 : 3
```

Calling a Function

- **name_of_the_function**(*argument*$_1$, *argument*$_2$, …, *argument*$_4$)

Defining a function

```
# this user defined function adds
def add_numbers(num1, num2):
    sum = num1 + num2
    return sum
```

Parameters

Calling a function

```
Shell
>>> res1 = add_numbers(1, 3)
>>> res1 = add_numbers(100, 5)
>>> res1 = add_numbers(50000, 123)
>>>
```

Arguments

- Function **calling name** should match function **definition name**
- Use *values*, *expression*, or *variables* to the **parameters** of the function
  - **arguments** should match **parameters**: one-to-one mapping
- When you call the function the execution gets transferred to the statements inside the function definition

Demo: calling with values

```python
# Author's name: Md Alimoor Reza
# Author's contact: md.reza@drake.edu
# Date: (September 7, 2021)
# Collaborator:
#       self


# this user defined function adds two numbers
def add_numbers(num1, num2):
    sum = num1 + num2
    print("add function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1+num2))
    return sum

def sub_numbers(num1, num2):
    sub = num1 - num2
    print("subtract function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1-num2))
    return sub

def mul_numbers(a,b):
    # Your task
    return
def div_numbers(a,b):
    # Your task
    return
```

Defining a function

Calling a function

Shell

```
Python 3.7.9 (bundled)
>>> %Run lec3_demo2.py
>>> sub = sub_numbers(10, 4)

  subtract function: called with num1=10 num2=4 and res=6

>>> print("result of subtraction from %d to %d is %d"%(10,4, sub))

  result of subtraction from 10 to 4 is 6
```

Demo: calling with variables

```python
# Author's name: Md Alimoor Reza
# Author's contact: md.reza@drake.edu
# Date: (September 7, 2021)
# Collaborator:
#       self


# this user defined function adds two numbers
def add_numbers(num1, num2):
    sum = num1 + num2
    print("add function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1+num2))
    return sum

def sub_numbers(num1, num2):
    sub = num1 - num2
    print("subtract function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1-num2))
    return sub

def mul_numbers(a,b):
    # Your task
    return
def div_numbers(a,b):
    # Your task
    return

```

Calling the same function with variables

Shell

```
>>> a = 10
>>> b = 4
>>> sub = sub_numbers(a, b)
  subtract function: called with num1=10 num2=4 and res=6
>>> print("result of subtraction from %d to %d is %d"%(a,b, sub))
  result of subtraction from 10 to 4 is 6
```

Demo: calling a function multiple times

```python
# Author's name: Md Alimoor Reza
# Author's contact: md.reza@drake.edu
# Date: (September 7, 2021)
# Collaborator:
#       self


# this user defined function adds two numbers
def add_numbers(num1, num2):
    sum = num1 + num2
    print("add function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1+num2))
    return sum

def sub_numbers(num1, num2):
    sub = num1 - num2
    print("subtract function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1-num2))
    return sub

```

Shell

```
  result of subtraction from 10 to 4 is 6

>>>

Python 3.7.9 (bundled)
>>> %Run lec3_demo2.py
>>> sub1 = sub_numbers(10, 4)

  subtract function: called with num1=10 num2=4 and res=6

>>> sub2 = sub_numbers(10, 5)

  subtract function: called with num1=10 num2=5 and res=5

>>> sub3 = sub_numbers(10, 6)

  subtract function: called with num1=10 num2=6 and res=4

>>>
```

Calling the function multiple times

Exercise 1: finish the rest and call them with various arguments

```python
# Author's name: Md Alimoor Reza
# Author's contact: md.reza@drake.edu
# Date: (September 7, 2021)
# Collaborator:
#       self


# this user defined function adds two numbers
def add_numbers(num1, num2):
    sum = num1 + num2
    print("add function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1+num2))
    return sum

def sub_numbers(num1, num2):
    sub = num1 - num2
    print("subtract function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1-num2))
    return sub

def mul_numbers(a,b):
    # Your task
    return
def div_numbers(a,b):
    # Your task
    return

```

Summary

- Takeaway from this lecture:
  - Functions are subroutine or helper algorithms and they can be invoked as many times as you like
  - <u>Defining a function</u> vs <u>calling a function</u>

- To do:
  - Start reading — Chapter 3

- Announcements:
  - **Next week (Tuesday, 09/13) there will be a quiz**
  - Paper based
  - Covering topics up to this week