

# 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**

```
33 time_sec = 60
34 temp_degree = 27
35
36 mile_to_kilometer = 1.609
37 price_in_dollars = 1500.89
```


Numbers

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

Textual data

# Recap: 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
```

# Recap: Comments

- Comments are notes explaining the functionality of your computer program (source code)
- Python comments are denoted with
  - `#` for a single line
  - triple quotes (either `'` or `“`) for multiple lines
- Other languages eg, C++ has different syntax

```
1 # Author's name: Md Alimoor  
2 # Author's contact: md.reza  
3 # Date: (September 1st, 2021)  
4 # Collaborator:  
5 #   Your partner's name  
6  
7 #print("Yay! this is my first python program in CS65!")  
8  
9  
10 #-----  
11 # add, multiplication, division
```

```
10 """  
11 Author's name: Md Alimoor Reza  
12 Author's contact: md.reza@drake.edu  
13 Date: (September 1st, 2021)  
14 Collaborator:  
15   Your partner's name  
16  
17 #print("Yay! this is my first python program in CS65!")  
18 """
```

# Recap: 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

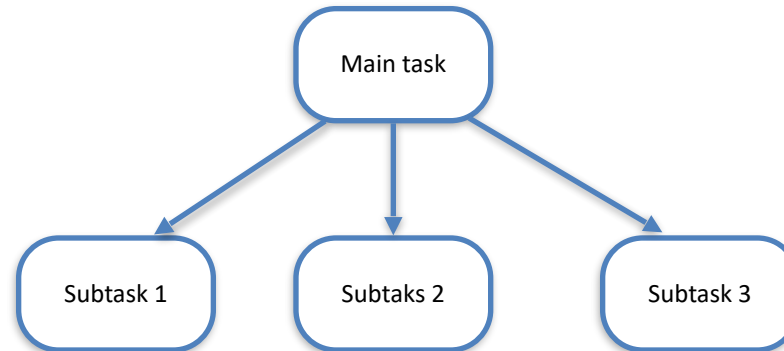
```
rect_a = input("enter the length of rectangle side a: ")  
print(rect_a)
```

# 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

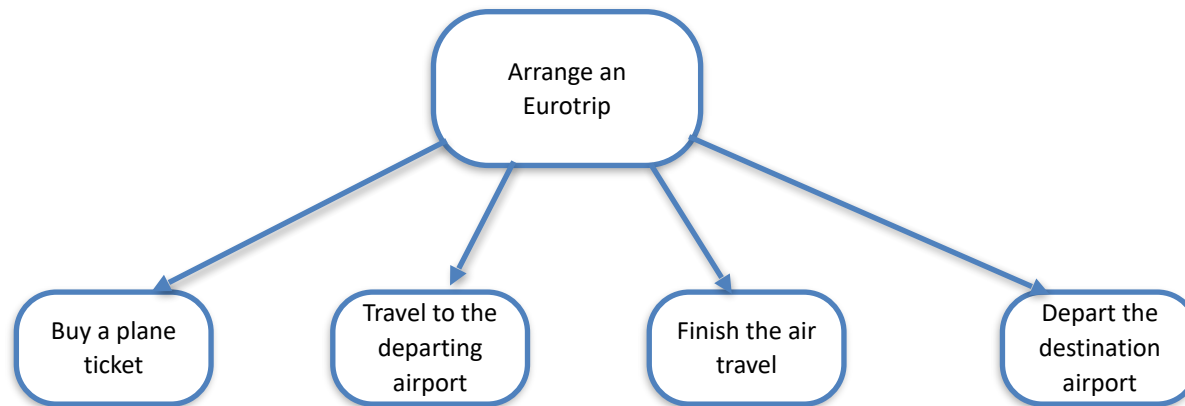
- **Function** is a sequence of statements that performs a specific task — also called a subroutine
- Decompose a bigger task with the help of several smaller subtasks





# 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 modular and manageable
    - Imagine, you have to write the same calculations over and over again eg 100 times!
  - Code for a subtask can be reusable
  - Individual member in a team can write different functions
  - Improves code readability

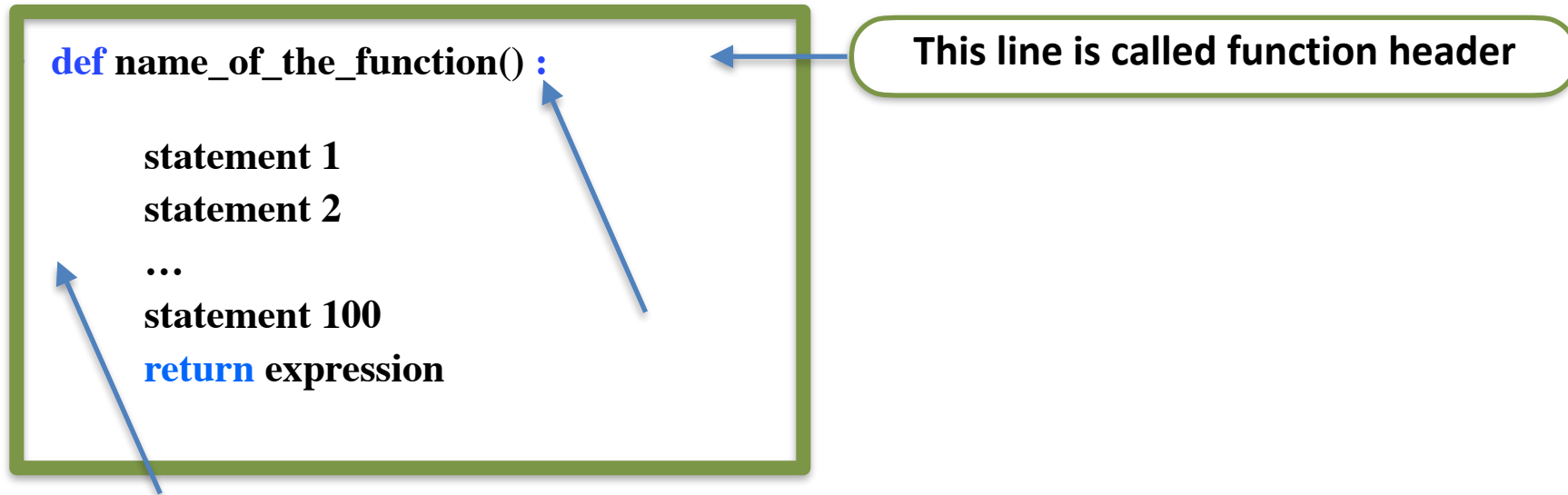
# Topics

- Functions — a new concept
  - User defined functions vs builtin functions
- A user defined function
  - define: what statements it will execute
  - call: invoke/execute the defined body

# 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

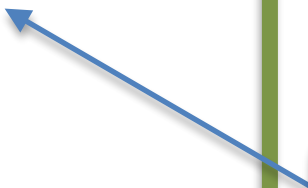
# Define a Function with no Parameters



- **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 variables used to exchange values during function call
  - Values are mapped to parameters each time the function is called
  - Parameters are not available outside the function

# Demo: user defined function example

```
1 # Author's name: Md Alimoor Reza
2 # Author's contact: md.reza@drake.edu
3 # Date: (September 7, 2021)
4 # Collaborator:
5 #     self
6
7
8 # this user defined function adds two numbers
9 def add_numbers(num1, num2):
10     sum = num1 + num2
11     return sum
12
13
```

Shell x

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

sum of 1 and 2 : 3
```

Watch out for these items!

# Calling a Function

- **name\_of\_the\_function**(*argument<sub>1</sub>*, *argument<sub>2</sub>*, ..., *argument<sub>4</sub>*)

## 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

```
1 # Author's name: Md Alimoor Reza
2 # Author's contact: md.reza@drake.edu
3 # Date: (September 7, 2021)
4 # Collaborator:
5 #     self
6
7
8 # this user defined function adds two numbers
9 def add_numbers(num1, num2):
10     sum = num1 + num2
11     print("add function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1+num2))
12     return sum
13
14 def sub_numbers(num1, num2):
15     sub = num1 - num2
16     print("subtract function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1-num2))
17     return sub
18
19 def mul_numbers(a,b):
20     # Your task
21     return
22 def div_numbers(a,b):
23     # Your task
24     return
25
```

**Defining a function**

Shell x

```
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
```

**Calling a function**

# Demo: calling with variables

```
1 # Author's name: Md Alimoor Reza
2 # Author's contact: md.reza@drake.edu
3 # Date: (September 7, 2021)
4 # Collaborator:
5 #     self
6
7
8 # this user defined function adds two numbers
9 def add_numbers(num1, num2):
10     sum = num1 + num2
11     print("add function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1+num2))
12     return sum
13
14 def sub_numbers(num1, num2):
15     sub = num1 - num2
16     print("subtract function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1-num2))
17     return sub
18
19 def mul_numbers(a,b):
20     # Your task
21     return
22 def div_numbers(a,b):
23     # Your task
24     return
25
```

Calling the same function  
with variables

Shell x

```
>>> 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

```
1 # Author's name: Md Alimoor Reza
2 # Author's contact: md.reza@drake.edu
3 # Date: (September 7, 2021)
4 # Collaborator:
5 #     self
6
7
8 # this user defined function adds two numbers
9 def add_numbers(num1, num2):
10     sum = num1 + num2
11     print("add function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1+num2))
12     return sum
13
14 def sub_numbers(num1, num2):
15     sub = num1 - num2
16     print("subtract function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1-num2))
17     return sub
18
```

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

```
1 # Author's name: Md Alimoor Reza
2 # Author's contact: md.reza@drake.edu
3 # Date: (September 7, 2021)
4 # Collaborator:
5 #     self
6
7
8 # this user defined function adds two numbers
9 def add_numbers(num1, num2):
10     sum = num1 + num2
11     print("add function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1+num2))
12     return sum
13
14 def sub_numbers(num1, num2):
15     sub = num1 - num2
16     print("subtract function: called with num1=%d num2=%d and res=%d"%(num1,num2, num1-num2))
17     return sub
18
19 def mul_numbers(a,b):
20     # Your task
21     return
22 def div_numbers(a,b):
23     # Your task
24     return
25
```

# Summary

- Takeaway from this lecture:
  - Functions are subroutine or helper algorithms and can be invoked as many times as we like
  - Defining a function vs calling a function
- To do:
  - Start reading — Chapter 3
- Announcements:
  - **Next week (Tuesday, 02/08) there will be a quiz**
  - Paper based
  - Covering topics up to this week