Chapter 6

Dr. Raza Ul Mustafa Khokhar
American University
Computer Science Department - CSC-148

Functions

- In Python, a function is a named sequence of statements that belong together. Their primary purpose is to help us organize programs into chunks that match how we think about the solution to the problem.

```
def name( parameters ):
    statements
```

The parameters specify what information, if any, you have to provide in order to use the new function. Another way to say this is that the parameters specify what the function needs to do its work.

A function to print some strings

```
def print_lyrics():
    print("I'm a lumberjack, and I'm okay.")
    print("I sleep all night and I work all day.")
```

```
>>> print(print_lyrics)
<function print_lyrics at 0xb7e99e9c>
>>> type(print_lyrics)
<class 'function'>
```

Calling a function

- The syntax for calling the new function is the same as for built-in functions:

```
print_lyrics()
```

Flow of execution

- To ensure that a function is defined before its first use, you have to know the order statements run in, which is called the flow of execution.
- Execution always begins at the first statement of the program. Statements are run one at a time, in order from top to bottom.
- Function definitions do not alter the flow of execution of the program, but remember that statements inside the function don’t run until the function is called.

Parameters and arguments

- Some of the functions we have seen require **arguments.**

```
def print_twice(bruce):
    print(bruce)
    print(bruce)
```

Inside the function, the arguments are assigned to variables called **parameters**

Variables and parameters are local

- When you create a variable inside a function, it is local, which means that it only exists inside the function. For example:

```
def cat_twice(part1, part2):
    cat = part1 + part2
    print_twice(cat)
```

```
line1 = 'Bing tiddle '
line2 = 'tiddle bang.'
cat_twice(line1, line2)
```

Function to return square root

```
def square(x):
    y = x * x
    return y

toSquare = 10
result = square(toSquare)
print("The result of", toSquare, "squared is", result)
```

Functions can Call Other Functions

```
def square(x):
    y = x * x
    return y
def sum_of_squares(x, y, z):
    a = square(x)
    b = square(y)
    c = square(z)
    return a + b + c
a = -5
b = 2
c = 10
result = sum_of_squares(a, b, c)
print(result)
```

Turtle example with functions

```
import turtle

def drawSquare(t, sz):
    """Make turtle t draw a square of with side sz."""

    for i in range(4):
        t.forward(sz)
        t.left(90)

wn = turtle.Screen()              # Set up the window and its attributes
wn.bgcolor("lightgreen")

alex = turtle.Turtle()            # create alex
drawSquare(alex, 50)             # Call the function to draw the square passing the
actual turtle and the actual side size
wn.exitonclick()
```

Tired of turtle? ... Let's change to another examples

Even Odd numbers?

Even or Odd Numbers

```
def is_even(number):
    if number % 2 == 0:
        return True
    else:
        return False

# Calling the function and using the return value
num = 6
if is_even(num):
    print(f"{num} is even.")
else:
    print(f"{num} is odd.")
```

Nested functions

```
def outer_function(x):
    def inner_function(y):
        return y * 2
    result = inner_function(x)
    return result

# Calling the nested functions
output = outer_function(3)
print("Output:", output)
```

*args as arugment

```
def average(*args):
    total = sum(args)
    count = len(args)
    avg = total / count
    return avg

print(average(2, 4, 6))     # Output: 4.0
print(average(10, 20, 30))  # Output: 20.0
```

Slides & Material

razaulmustafa.us/cs148/