Web Application Development using Python

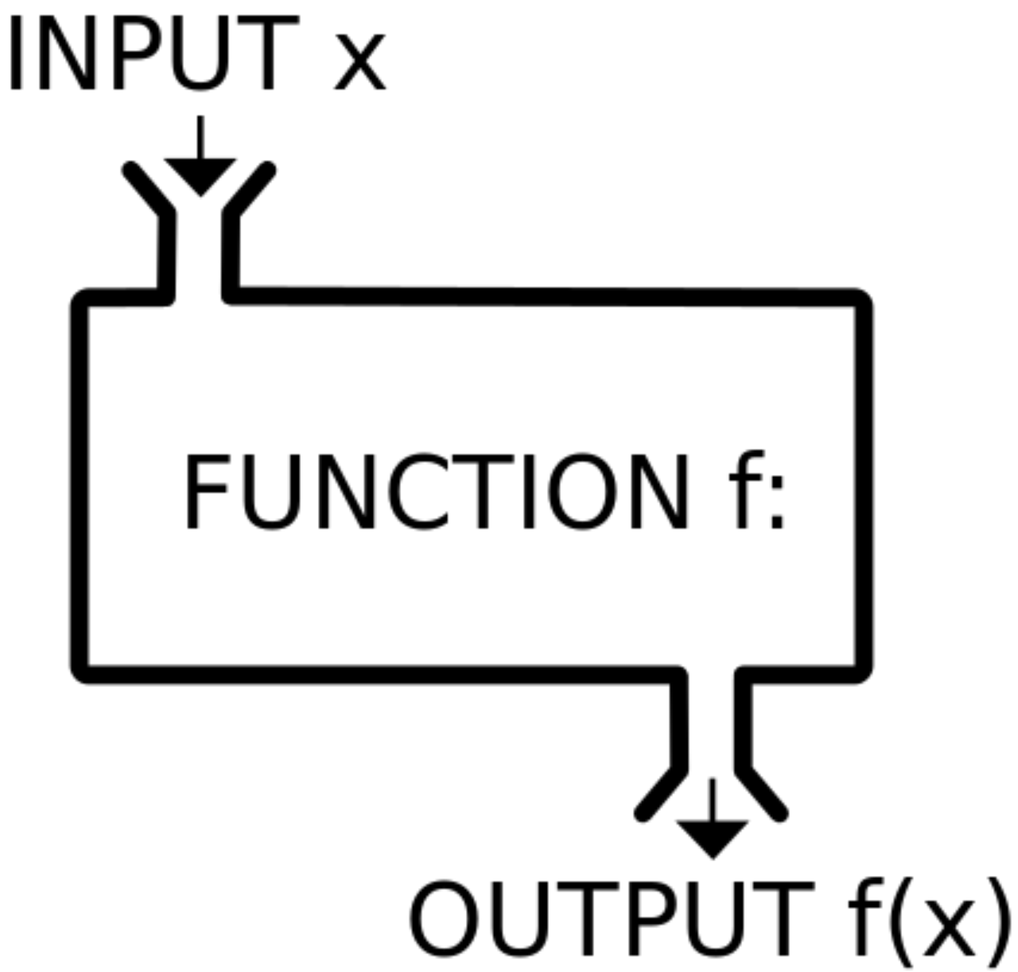
Introduction to Functions

Prepared by George Khoury

Outline

- **Function Syntax**
  - Creating a function
  - Calling a function
- **Function Arguments**
  - Passing Arguments
  - Types of Arguments
- Return Value
- Recursion

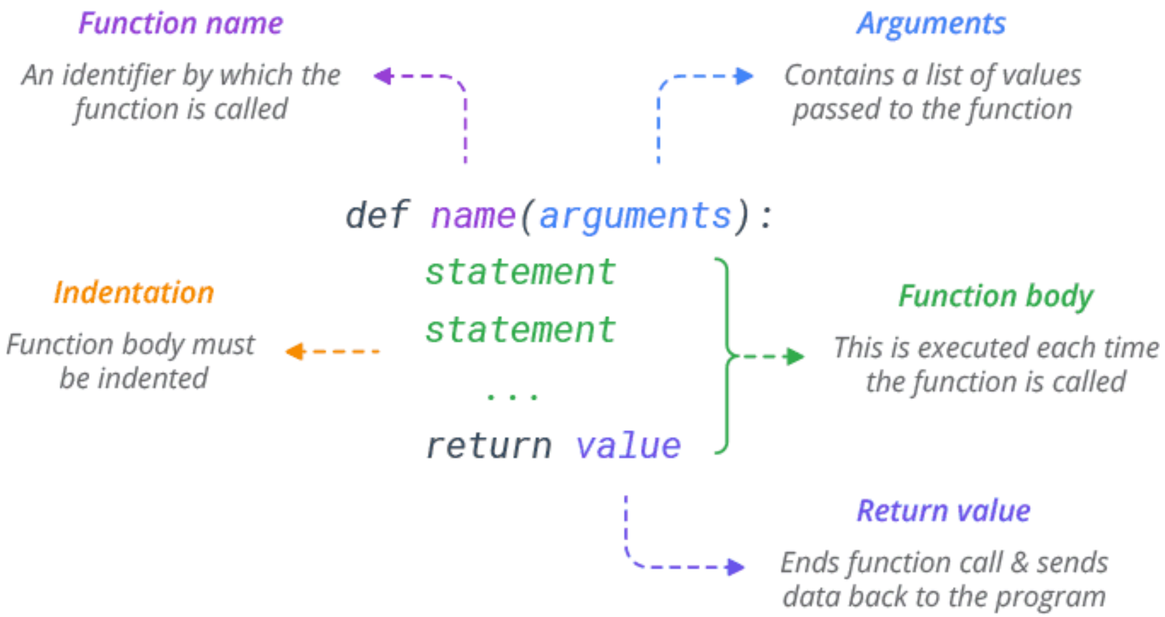
Function Syntax



Function Syntax

W3/S2/function_basics.py

- Functions are the first step to code reuse.
- They allow you to define a reusable block of code that can be used repeatedly in a program.
- Python provides several built-in functions such as `print()`, `len()` or `type()`.
- You can also define your own functions to use within your programs.



Creating a Function

W3/S2/function_basics.py

- To define a Python function, use def keyword.
- Here's the simplest possible function that prints 'Hello, World!' on the screen.

```
def hello():
    print('Hello, World!')
```

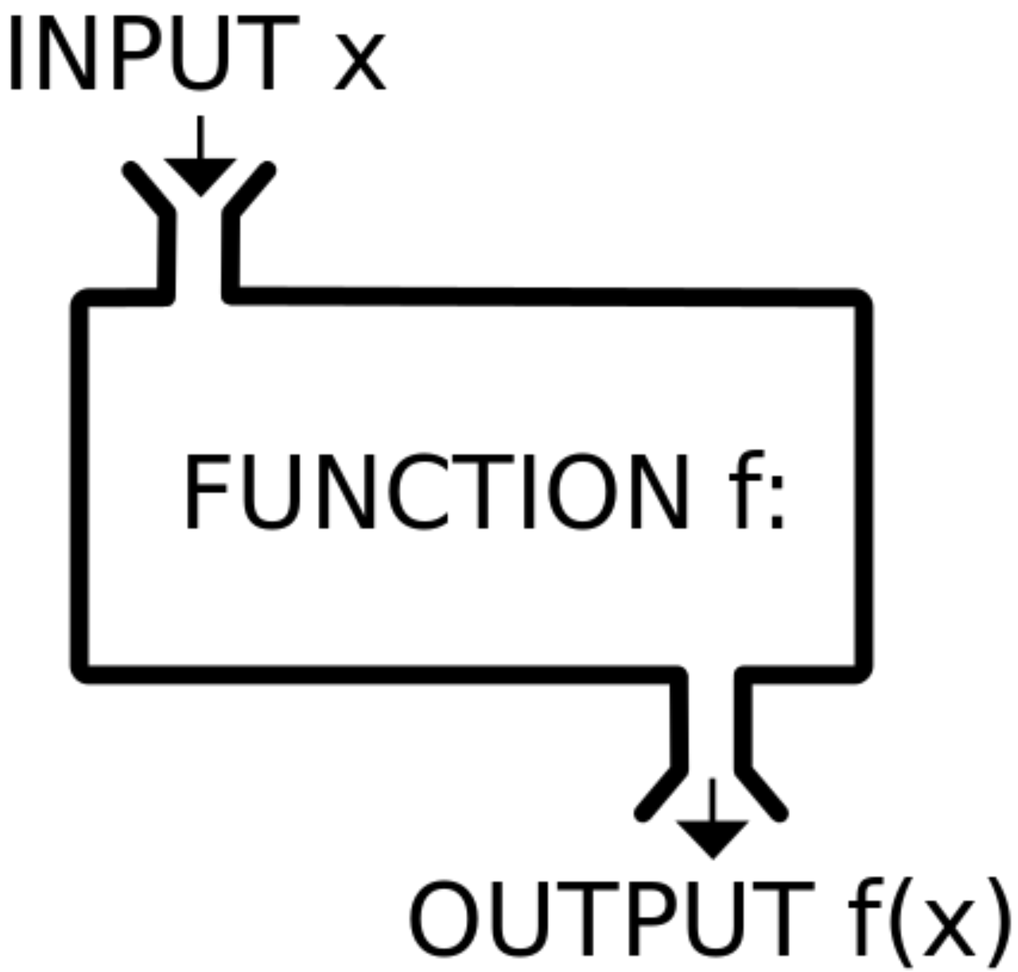
Calling a Function

W3/S2/function_basics.py

- The **`def`** statement only creates a function but does not call it.
- After the **`def`** has run, you can can call (run) the function by adding parentheses after the function's name.

```
def hello():
    print('Hello, World!')

hello()
# Prints Hello, World!
```

Function Arguments



Passing Arguments

W3/S2/function_basics.py

- You can send information to a function by passing values, known as arguments.
- Arguments are declared after the function name in parentheses.
- When you call a function with arguments, the values of those arguments are copied to their corresponding parameters inside the function.
- You can send as many arguments as you like, separated by commas.

```
# Pass single argument to a function
def hello(name):
    print('Hello,', name)

hello('Bob')
# Prints Hello, Bob

hello('Sam')
# Prints Hello, Sam
```

```
# Pass two arguments
def print_job(name, job):
    print(name, 'is a', job)

print_job('Bob', 'developer')
# Prints Bob is a developer
```

Types of Arguments

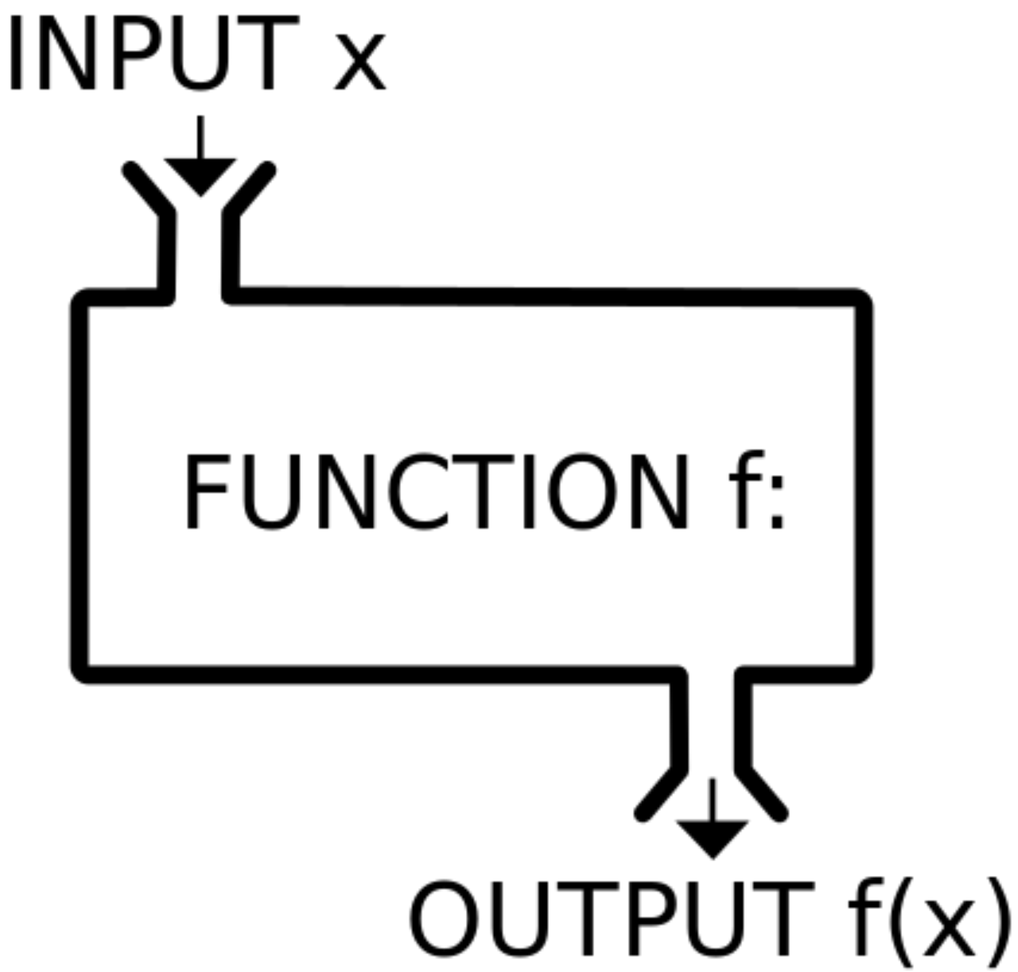
W3/S2/function_arguments.py

- Python supports multiple types of arguments in the function definition.
  - Positional Arguments
  - Keyword Arguments
  - Default Arguments
  - Variable Length Positional Arguments (*args)
  - Variable Length Keyword Arguments (**kwargs)

```
# Pass single argument to a function
def hello(name):
    print('Hello,', name)

hello('Bob')
# Prints Hello, Bob

hello('Sam')
# Prints Hello, Sam
```

```
# Pass two arguments
def print_job(name, job):
    print(name, 'is a', job)

print_job('Bob', 'developer')
# Prints Bob is a developer
```

Return Value



Return Value

W3/S2/function_return.py

- To return a value from a function, simply use a `return` statement.
- Once a return statement is executed, **nothing else in the function body is executed.**
- **Remember!**
  - A python function always returns a value. So, if you do not include any return statement, it automatically returns None.

```
# Return sum of two values
def sum(a, b):
    return a + b

x = sum(3, 4)
print(x)
# Prints 7
```

Return Value

W3/S2/function_return.py

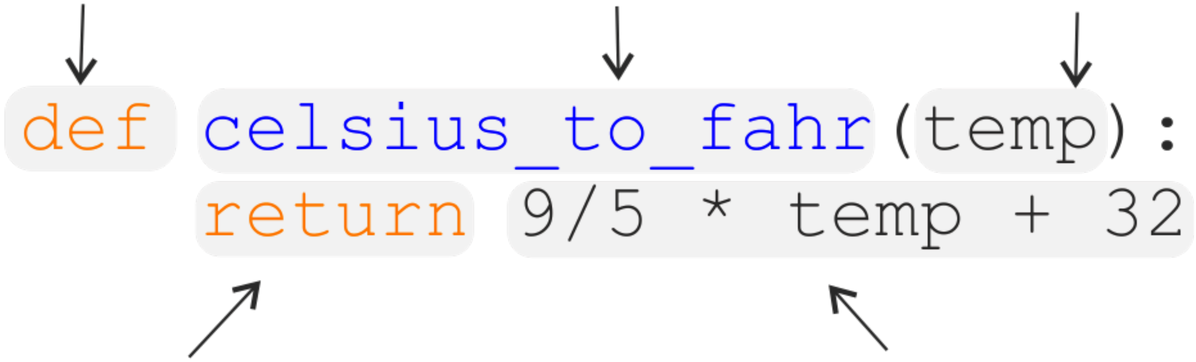
- Python has the ability to return multiple values.
- You can do this by separating return values with a comma.
- When you return multiple values, Python packs them in a single tuple and returns it.
- You can then use multiple assignment to unpack the parts of the returned tuple.

```
# Return addition and subtraction in
a tuple
def do_math(a, b):
    return a+b, a-b

result = do_math(3, 2)

print(result)
# Prints (5, 1)
```

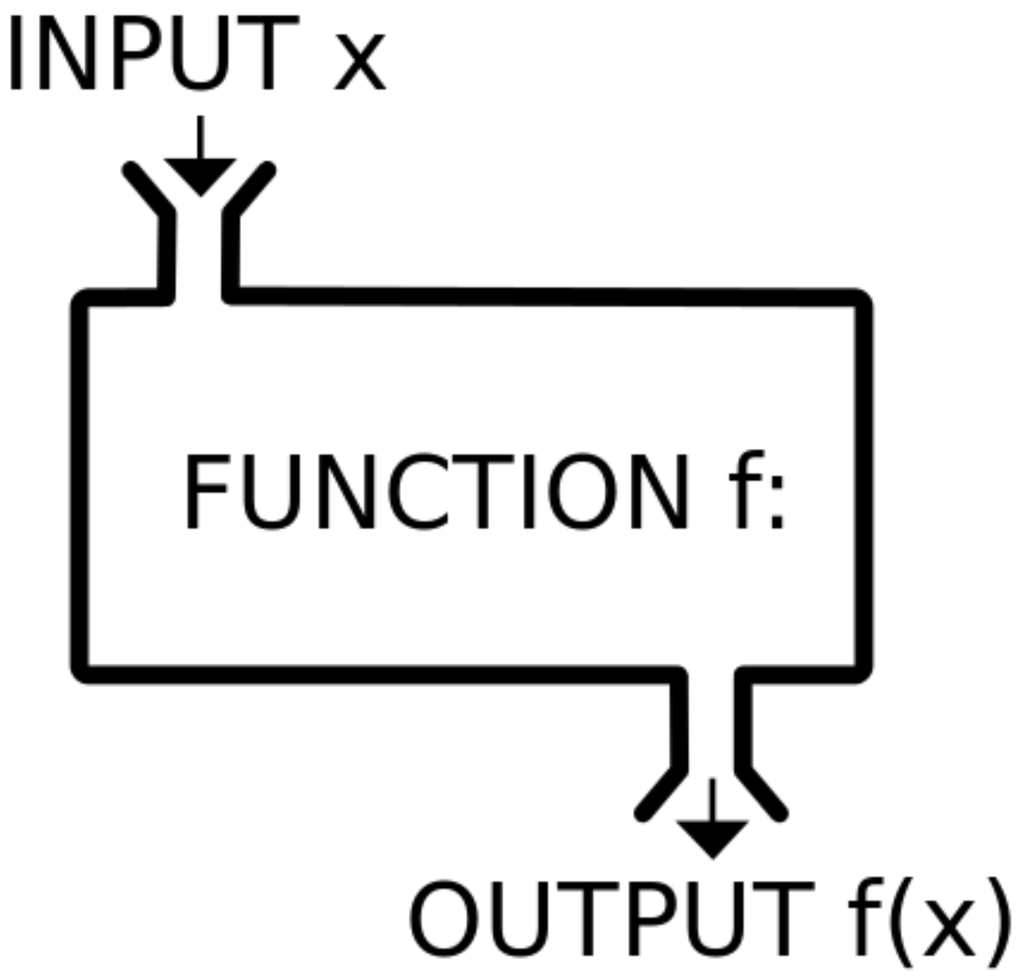
```
# Unpack returned tuple
def do_math(a, b):
    return a+b, a-b

add, sub = do_math(3, 2)

print(add)
# Prints 5
print(sub)
# Prints 1
```

Functions

- What is a **function signature**?
  - Which keyword is used to define the function?
  - What is the name of the function?
  - How many arguments are being passed to this function? What are the names?
- What is a **function body**?
  - How many statements does it contain?
  - What is the return value?



Recursion



Recursion

W3/S2/function_recursion.py

- A recursive function is a function that calls itself and repeats its behavior until some condition is met to return a result.
- We can use recursive functions, for example, to implement a factorial function.

```
def fact(n):
    if n == 1:
        return 1
    elif n < 1:
        print('Choose a number greater than zero.')
    else:
        return n* fact(n-1)

print(fact(5))
# Prints 120
```

Learning Resources

- https://docs.python.org/3.7/tutorial/controlflow.html#defining-functions
- https://docs.python.org/3.7/tutorial/controlflow.html#more-on-defining-functions
- https://docs.python.org/3/library/functions.html
- https://docs.python.org/3.8/tutorial/controlflow.html#documentation-strings
- https://docs.python.org/3.8/tutorial/controlflow.html#function-annotations