

Functions

Stored (and reused) Steps

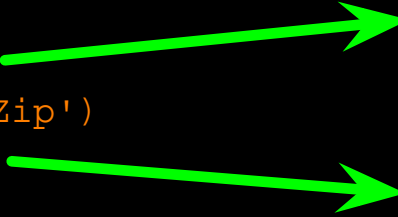
Program:

```
def thing():  
    print('Hello')  
    print('Fun')
```

```
thing()  
print('Zip')  
thing()
```

Output:

```
Hello  
Fun  
Zip  
Hello  
Fun
```



We call these reusable pieces of code “functions”

Python Functions

- There are two kinds of functions in Python.
 - **Built-in functions** that are provided as part of Python - print(), input(), type(), float(), int() ...
 - **Functions that we define ourselves** and then use
- We treat function names as “new” **reserved words** (i.e., we avoid them as variable names)

Function Definition

- In Python a **function** is some reusable code that takes **arguments**(s) as input, does some computation, and then returns a result or results
- We define a **function** using the **def** reserved word
- We call/invoke the **function** by using the function name, parentheses, and **arguments** in an expression

Type Conversions

- When you put an integer and floating point in an expression, the integer is **implicitly** converted to a float
- You can control this with the built-in functions `int()` and `float()`

```
>>> print(float(99) / 100)
0.99
>>> i = 42
>>> type(i)
<class 'int'>
>>> f = float(i)
>>> print(f)
42.0
>>> type(f)
<class 'float'>
```

String Conversions

- You can also use `int()` and `float()` to convert between strings and integers
- You will get an `error` if the string does not contain numeric characters

```
>>> sval = '123'
>>> type(sval)
<class 'str'>
>>> print(sval + 1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str
(not "int") to str
>>> ival = int(sval)
>>> type(ival)
<class 'int'>
>>> print(ival + 1)
124
>>> nsval = 'hello bob'
>>> niv = int(nsval)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int()
```

Functions of Our Own...

Building our Own Functions

- We create a new function using the **def** keyword followed by optional parameters in parentheses
- We indent the body of the function
- This **defines** the function but **does not** execute the body of the function

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

Definitions and Uses

- Once we have **defined** a function, we can **call** (or **invoke**) it as many times as we like
- This is the **store** and **reuse** pattern

```
x = 5
```

```
print('Hello')
```

```
def print_lyrics():
```

```
    print("I'm a lumberjack, and I'm okay.")
```

```
    print('I sleep all night and I work all day.')
```

```
print('Yo')
```

```
print_lyrics()
```

```
x = x + 2
```

```
print(x)
```

Hello

Yo

I'm a lumberjack, and I'm okay.

I sleep all night and I work all day.

7

Arguments

- An **argument** is a value we pass into the **function** as its **input** when we call the function
- We use **arguments** so we can direct the **function** to do different kinds of work when we call it at **different** times
- We put the **arguments** in parentheses after the **name** of the function

```
length = len('Hello world')
```

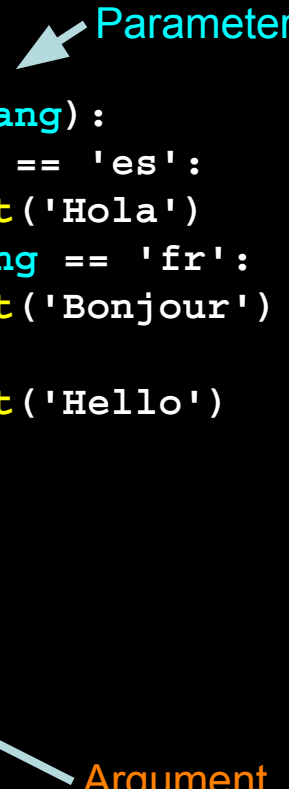


Argument

Parameters

A **parameter** is a variable which we use in the **function definition**.

```
>>> def greet(lang):  
...     if lang == 'es':  
...         print('Hola')  
...     elif lang == 'fr':  
...         print('Bonjour')  
...     else:  
...         print('Hello')  
...  
>>> greet('en')  
Hello  
>>> greet('es')  
Hola  
>>> greet('fr')  
Bonjour  
>>>
```



Return Values

Often a function will take its arguments, do some computation, and **return** a value to be used as the output of the function. The **return** keyword is used for this. It ends the **function** execution and “sends back” the **result** of the **function**

```
def greet():  
    return "Hello"
```

```
Hello Glenn  
Hello Sally
```

```
print(greet(), "Glenn")  
print(greet(), "Sally")
```

Multiple Parameters / Arguments

- We can define more than one **parameter** in the **function definition**
- We simply add more **arguments** when we call the **function**
- We match the number and order of arguments and parameters

```
def addtwo(a, b):  
    added = a + b  
    return added
```

```
x = addtwo(3, 5)  
print(x)
```

8



Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance (www.dr-chuck.com) of the University of Michigan School of Information and open.umich.edu and made available under a Creative Commons Attribution 4.0 License. Please maintain this last slide in all copies of the document to comply with the attribution requirements of the license. If you make a change, feel free to add your name and organization to the list of contributors on this page as you republish the materials.

Initial Development: Charles Severance, University of Michigan School of Information

Adapted by Minh Tran, University of Chicago, Department of Computer Science for use in the pre-college course Creating Computer Games for Learning (Summer 2026).