



# Summer Coding School 2023

Aug 26, 2023

**Taehee Jeong, Ph.D.**

# Data type

- Integer
- Floating point
- String
- Boolean

# Data structure

- List
- Set
- Tuples
- Dictionary

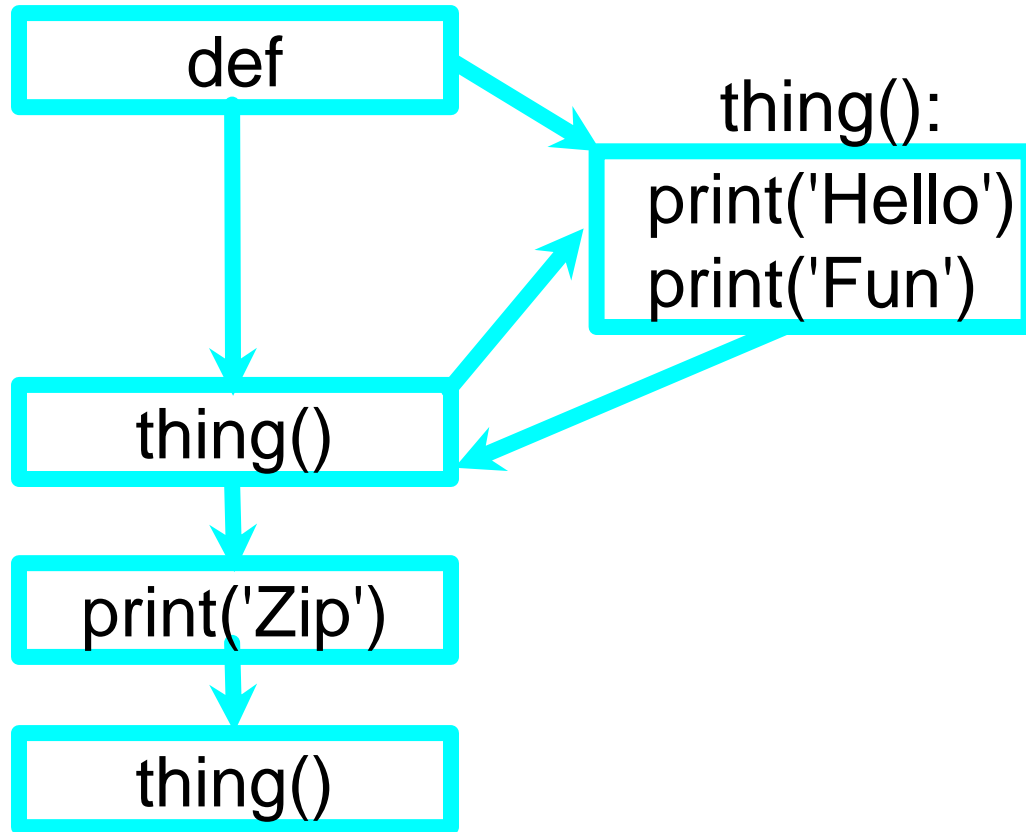
# Control Flows

- Conditional: IF statement
- While Loop
- For Loop

# Today's Agenda

- Functions
- Class

# 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 the built-in function names as “new” reserved words (i.e., we avoid them as variable names)

# Built-in function: max()

```
>>> big = max('Hello world')
```

```
>>> print(big)
```

w

*A function is some stored code that we use. A function takes some input and produces an output.*

'Hello world'  
(a string)

max()  
function

'w'  
(a string)



# 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

# 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

# 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

```
big = max('Hello world')
```



Argument

# Parameters

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

It is a “handle” that allows the code in the function to access the arguments for a particular function invocation.

```
>>> 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 value of the function call in the calling expression.

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

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

```
Hello Glenn  
Hello Sally
```

# Return Value

A “fruitful” function is one that produces a result (or return value)

The return statement ends the function execution and “sends back” the result of the function

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

# 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
```



# Void (non-fruitful) Functions

- When a function does not return a value, we call it a “void” function
- Functions that return values are “fruitful” functions
- Void functions are “not fruitful”

# To function or not to function...

- Organize your code into “paragraphs” - capture a complete thought and “name it”
- Don’t repeat yourself - make it work once and then reuse it
- If something gets too long or complex, break it up into logical chunks and put those chunks in functions
- Make a library of common stuff that you do over and over - perhaps share this with your friends...

A Class

class is a reserved word

This is the template for making PartyAnimal objects

```
class PartyAnimal:
    x = 0

    def party(self) :
        self.x = self.x + 1
        print("So far",self.x)
```

Each PartyAnimal object has a bit of data

```
an = PartyAnimal()
```

Construct a PartyAnimal object and store in an

```
an.party()
an.party()
an.party()
```

PartyAnimal.party(an)

Tell the an object to run the party() code within it

```
class PartyAnimal:
    x = 0

    def party(self) :
        self.x = self.x + 1
        print("So far",self.x)

an = PartyAnimal()

an.party()
an.party()
an.party()
```

So far 1  
So far 2  
So far 3

# Find Capabilities

- The `dir()` command lists capabilities
- Ignore the ones with underscores - these are used by Python itself
- The rest are real operations that the object can perform
- It is like `type()` - it tells us something *\*about\** a variable

```
>>> y = list()
>>> type(y)
<class 'list'>
>>> dir(y)
['__add__', '__class__',
 '__contains__', '__delattr__',
 '__delitem__', '__delslice__',
 '__doc__', ... '__setitem__',
 '__setslice__', '__str__', 'append',
 'clear', 'copy', 'count', 'extend',
 'index', 'insert', 'pop', 'remove',
 'reverse', 'sort']
```

```
class PartyAnimal:
    x = 0

    def party(self) :
        self.x = self.x + 1
        print("So far",self.x)

an = PartyAnimal()

print("Type", type(an))
print("Dir ", dir(an))
```

We can use `dir()` to find the “capabilities” of our newly created class.

# Try dir() with a String

```
>>> x = 'Hello there'
>>> dir(x)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
'__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
'__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__',
'__lt__', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__str__',
'capitalize', 'center', 'count', 'decode', 'encode', 'endswith',
'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit',
rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
'translate', 'upper', 'zfill']
```



# Acknowledgements / Contributions



These slides are Copyright 2010- Charles R. Severance ([www.dr-chuck.com](http://www.dr-chuck.com)) of the University of Michigan School of Information 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

Modification: Taehee Jeong, San Jose State University