Introduction to Software Systems (CS6.201)

Summer 2021, IIIT Hyderabad

08 July, Thursday (Lecture 13) – Python 4

Taught by Abhinav Gupta

Try-Catch Blocks

The code in the `try` block is executed; in case of any error in this code, the code in the `except` block is run.

Lambdas

Lambda syntax allows us to write anonymous functions. For example, if one wishes to sort a dictionary according to its values, one way would be

```
d = {'apple': 18, 'orange': 20, 'banana': 5, 'tomato': 1}
d = sorted(d.items(), key = lambda x : x[1])
```

where the `key` argument is an anonymous function from items to values.

Extended Arguments

Functions can be defined to take an indeterminate number of arguments using the following syntax:

```
def hypervolume(*dims):
    a = 1
    for d in dims:
        a *= d
    print(a)
```

Now, the function `hypervolume` calculates the product of all its arguments (no matter how many there are). Within the function, `dims` behaves like a list of the arguments.

When the parameter is prefixed with `**`, the arguments can be named. For instance,

```
def tag(name, **kwargs):
    within = name
    for attr in kwargs.items:
        within += ' ' + attr[0] + '=\"' + attr[1] + '\"'
    within = '<' + within + '>'
    return within
```

This function can be used to construct an HTML tag – for example, `tag('img', src='iiit.jpg', alt='college')` would return `<img src="iiit.jpg" alt="college">`.

Higher-Order Functions

Functions can be assigned and passed just like variables.

```
def greeting(name):
    print('Hello', name)

g = greeting
```

Now `g` is a function that does exactly the same thing as `greeting`.

Similarly,

```
def after(second, func):
    time.sleep(second)
    func()
```

would wait for `second` seconds and then run `func`, which is also passed as an argument.

A closure is a function that returns a function. For example,

```
def add(x,y):
    def add_closure():
        print('Adding {} + {}'.format(x,y)
        return x+y
    return add_closure
```

Now, `a = add(2,3)` assigns a function to `a`. If we try to print `a`, we will not be able to; we must run `a()` in order to get the correct output, which is

```
Adding 2 + 3
5
```