Python Notes 3

Lambda functions

A lambda function is a small (one line) anonymous function that is defined without a name. A lambda function can take any number of arguments, but can only have one expression. While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword.

Ex.

```
# a lambda function that adds 10 to the input argument
f = lambda x: x+10
val1 = f(5)
val2 = f(100)
print(val1, val2)

# a lambda function that multiplies two input arguments and returns the result
f = lambda x,y: x*y
val3 = f(2,10)
val4 = f(7,5)
print(val3, val4)
```

Custom sorting using a lambda function as key parameter

The key function transforms each element before sorting.

```
points2D = [(1, 9), (4, 1), (5, -3), (10, 2)]
sorted_by_y = sorted(points2D, key= lambda x: x[1])
print(sorted_by_y)

mylist = [- 1, -4, -2, -3, 1, 2, 3, 4]
sorted_by_abs = sorted(mylist, key= lambda x: abs(x))
print(sorted_by_abs)
```

Python Functions

In Python, a function is a group of related statements that performs a specific task.

Arguments and parameters

- Parameters are the variables that are defined or used inside parentheses while defining a function
- Arguments are the value passed for these parameters while calling a function

```
def print_name(name): # name is the parameter
    print(name)

print_name('Asem') # 'Asem' is the argument
```

Default arguments

```
# default arguments
def test_function(a, b, c, d=4):
    print(a, b, c, d)

test_function(1, 2, 3, 4)
test_function(1, b=2, c=3, d=100)

def test_function(a, b=2, c, d=4):
    print(a, b, c, d)

# not allowed: default arguments must be at the end
# def test_function(a, b=2, c, d=4):
#     print(a, b, c, d)
```

Variable-length arguments (* *args and ***kwargs)

- If you mark a parameter with one asterisk (*), you can pass any number of positional arguments to your function (Typically called *args)
- If you mark a parameter with two asterisks (***), you can pass any number of keyword arguments to this function (Typically called **kwargs).

```
def test_function(a, b, *args, **kwargs):
    print(a, b)
    for arg in args:
        print(arg)
    for kwarg in kwargs:
        print(kwarg, kwargs[kwarg])

# 3, 4, 5 are combined into args
# six and seven are combined into kwargs
test_function(1, 2, 3, 4, 5, six=6, seven=7)
print()

# omitting of args or kwargs is also possible
test_function(1, 2, three=3)
```