FUNCTION

PREPARED BY: LUIS MEING

OBJECTIVES:

01 Definition

02 Built-in Functions

03 User-defined Functions

04 Nested Functions

05 Variable Scope

06 Compare and Contrast

1.1 Definition

Functions

- are 'groups of code blocks' with a name

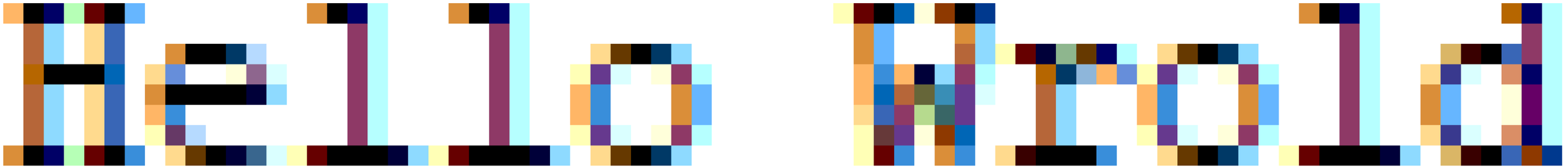
1.1 Definition

Functions

```
1  class Demo{
2      public static void main(String args[]){
3          System.out.println("Hello Wrold");
4      }
5  }
```

1.1 Definition

Functions



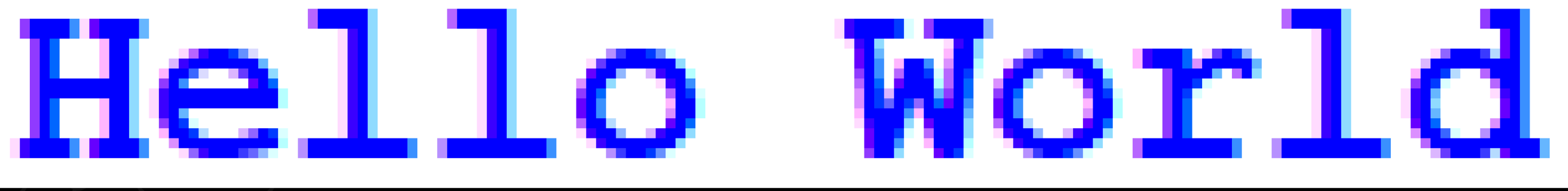
1.1 Definition

Functions

```
print("Hello World")
```

1.1 Definition

Functions



Functions vs. Methods

- How many are the functions?

- How many are the methods?

1.1 Definition

Functions vs. Methods

- A function is a set of instructions or procedures to perform a specific task

- A method is a set of instructions that are associated with an object.

Built-in Functions

- functions that are part of the 'libraries' you downloaded when you started to program in that language

Built-in Functions

```
System.out.println("Hello Wrold");
```

```
print("Hello World")
```

User-defined Functions

- take note to not use keywords to name your user-defined functions

User-defined Functions Syntax

- In Python, create as:

```
def <function_name>(<parameters>):
        <content>
```

User-defined Functions Syntax

```
def hello_world():
    print("Hello World")
```

User-defined Functions Syntax

- In Python, refer as :

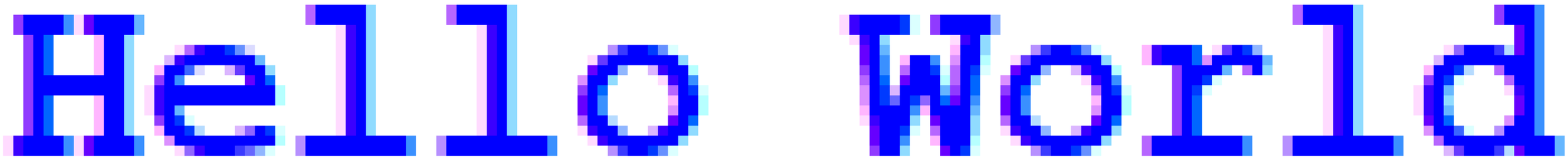
<function_name>(<arguments>)

User-defined Functions Syntax

```
def hello_world():
    print("Hello World")


hello_world()
```

User-defined Functions Syntax



Arguments vs. Parameters

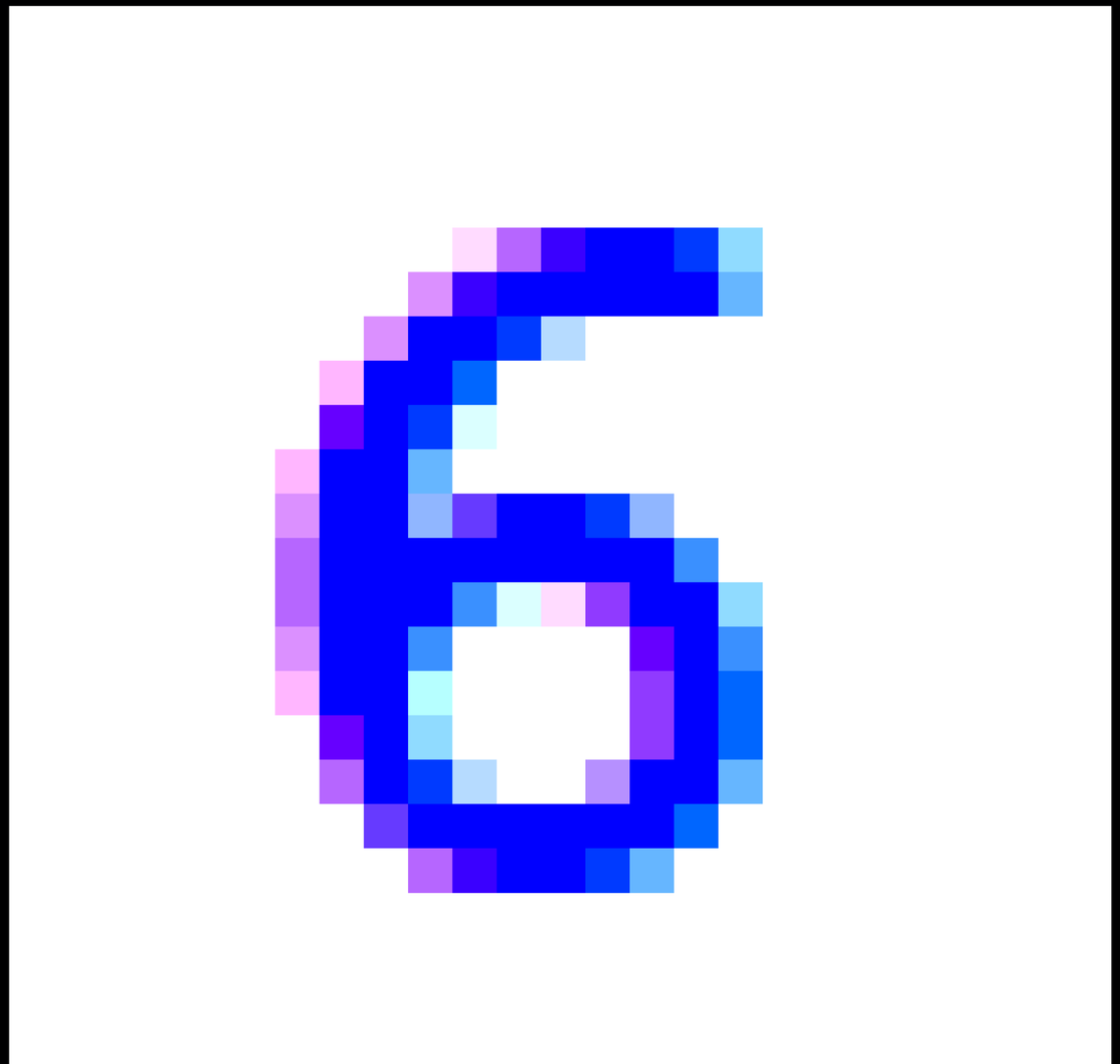
- arguments are values put inside the parentheses when referring to a function

- parameters are the variables put inside the parentheses when creating a function

Arguments vs. Parameters

```
1 def multiply(x, y):
2     return x*y
3
4 a = 2
5 b = 3
6 print( multiply(a, b) )
```

Arguments vs. Parameters



Return

- refers to what the function will give back as a value when the function is used
- is optional, uses the ‘return’ Keyword

- unlike other languages, you do not need to explicitly declare a return type of the function in Python

Functions with an Arbitrary Number of Arguments

- sometimes, there may not be argument that will be passed to the function
- in Python, we use the '*' symbol as a prefix

Functions with an Arbitrary Number of Arguments

```
1 def multi_args(*args):
2     for i in args:
3         print(i)
```

Functions with an Arbitrary Number of Arguments

- we can pass primitives and non-primitive values

Functions with an Arbitrary Number of Arguments

```
1 def multi_args(*args):
2     for i in args:
3         print(i)
4
5
6 sample_list = [5, 8, 10, 12, 15]
7 multi_args(sample_list)
8 multi_args(*sample_list)
```

Functions with an Arbitrary Number of Arguments

```
[5, 8, 10, 12, 15]
5
8
10
12
15
```

Functions with an Arbitrary Number of Arguments

- when passing mutable structures, take note:

- Mutating a parameter will mutate the argument
- reassigning the parameter won’t reassign the argument

Functions with an Arbitrary Number of Arguments

```
1 def sample(x):
2     x[0] = 10
3     x = [20, 30, 40]
4     x[1] = 15
5     print(x)
6
7 y = [5, 6, 7]
8 sample(y)
9 print(y)
```

Functions with an Arbitrary Number of Arguments

```
[20, 15, 40]
[10, 6, 7]
```

User-defined Functions Syntax

- In Java, create as:

```
<access_modifier> <static> <return_type><method name>([parameters]){
        <method body>
}
```

User-defined Functions Syntax

```
<access_modifier> <static> <return_type><method name>([parameters]){
    <method body>
}
```

- access modifier
  - either public or private
  - by default is public

User-defined Functions Syntax

```
<access_modifier> <static> <return_type><method name>([parameters]){
    <method body>
}
```

- static
  - will determine if the method is 'static' or an 'instance' method

User-defined Functions Syntax

```
<access_modifier> <static> <return_type><method name>([parameters]){
    <method body>
}
```

- return type
  - refers to what the function will give back as a value when called
  - uses the 'return' keyword is uses a data type

User-defined Functions Syntax

```
public static void main(String args[]){
    System.out.println("Hello Wrold");
}
```

User-defined Functions Syntax

```
1   package com.mycompany.demonstration;
2
3   public class Demonstration {
4       static String hello(){
5           return "Hello Wrold";
6       }
7
8       public static void main(String args[]){
9           hello();
10      }
11  }
```

User-defined Functions Syntax

```
------------------------------------------------------------------------

BUILD SUCCESS

------------------------------------------------------------------------

Total time:  3.959 s
Finished at: 2022-11-11T08:03:14+08:00

------------------------------------------------------------------------
```

User-defined Functions Syntax

```
1   package com.mycompany.demonstration;
2
3   public class Demonstration {
4       static String hello(){
5           return "Hello Wrold";
6       }
7
8       public static void main(String args[]){
9           System.out.println(hello());
10      }
11  }
```

User-defined Functions Syntax

```
Building Demonstration 1.0-SNAPSHOT
--------------------------------[ jar ]---------

--- exec-maven-plugin:3.0.0:exec (default-cli)
Hello Wrold
------------------------------------------------
BUILD SUCCESS
------------------------------------------------
Total time:  4.208 s
Finished at: 2022-11-11T08:05:42+08:00
------------------------------------------------
```

User-defined Functions Syntax

```
1   package com.mycompany.demonstration;
2
3   public class Demonstration {
4       static void hello(){
5           System.out.println("Hello Wrold");
6       }
7
8       public static void main(String args[]){
9           hello();
10      }
11  }
```

User-defined Functions Syntax

```
Building Demonstration 1.0-SNAPSHOT
--------------------------------[ jar ]---------

--- exec-maven-plugin:3.0.0:exec (default-cli)
Hello Wrold
------------------------------------------------
BUILD SUCCESS
------------------------------------------------
Total time:  2.189 s
Finished at: 2022-11-11T08:08:17+08:00
------------------------------------------------
```

User-defined Functions Syntax

```
1   package com.mycompany.demonstration;
2
3   public class Demonstration {
4
5       public static void main(String args[]){
6           char hello[] = {'h','e','l','l','o'};
7
8           System.out.println(hello.length);
9           System.out.println(hello);
10      }
11  }
```

User-defined Functions Syntax

```
Building Demonstration 1.0-SNAPSHOT
--------------------------------[ jar ]---------------------------------

--- exec-maven-plugin:3.0.0:exec (default-cli) @ Demonstration ---
5
hello
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  2.487 s
Finished at: 2022-11-11T08:19:36+08:00
------------------------------------------------------------------------
```

Nested Functions

- you can place functions inside another function
- this is possible in all programming languages that follow OOP
- take note that parameters passed to the outer function are also passed to the inner function/s

Nested Functions

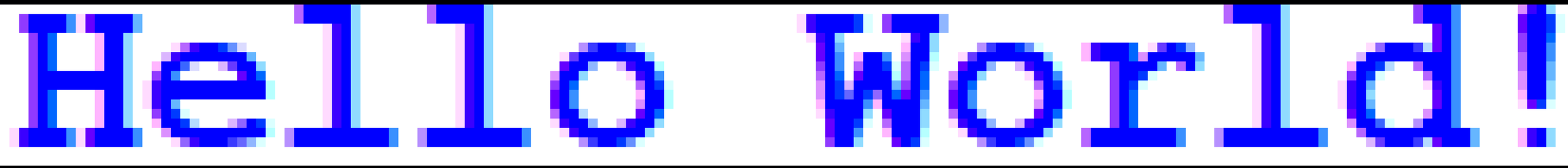
- you can place functions inside another function
- this is possible in all programming languages that follow OOP
- take note that parameters passed to the outer function are also passed to the inner function/s

4.1 Nested Functions

```
def outer_func(who):
    def inner_func():
        print("Hello", who)
    inner_func()


outer_func("World!")
```

Nested Functions



4.1

```
1  package com.mycompany.demonstration;
2
3  public class Demonstration {
4      static void Foo() {
5          class Local {
6
7              void fun() {
8                  System.out.println("Hello");
9              }
10         }
11         new Local().fun();
12     }
13
14     public static void main(String[] args) {
15         Foo();
16     }
17 }
```

Nested Functions

```
-----------------------< com.mycompany:Demonstration >-----------------------
Building Demonstration 1.0-SNAPSHOT
--------------------------------[ jar ]---------------------------------

--- exec-maven-plugin:3.0.0:exec (default-cli) @ Demonstration ---
Hello
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  1.771 s
Finished at: 2022-11-11T14:05:14+08:00
------------------------------------------------------------------------
```

Variable Scope

- Determines where in the program the variable is accessible
- Determines the lifetime (how long the variable can exist in memory)

- Determined by placement of the variable or where the variable is declared

Variable Scope

- Determines where in the program the variable is accessible
- Determines the lifetime (how long the variable can exist in memory)

- Determined by placement of the variable or where the variable is declared

Variable Scope

```
1   package com.mycompany.demonstration;
2
3   public class Demonstration {
4       public static void main(String args[]){
5           for(int i=0;i<5;i++){
6               System.out.println(i);
7           }
8           System.out.println(i);
9       }
10  }
```

Variable Scope

```
-------------------< com.mycompany:Demonstration >--------------------
Building Demonstration 1.0-SNAPSHOT
--------------------------------[ jar ]---------------------------------

--- exec-maven-plugin:3.0.0:exec (default-cli) @ Demonstration ---
Exception in thread "main" java.lang.RuntimeException: Uncompilable code
- cannot find symbol
  symbol:   variable i
  location: class com.mycompany.demonstration.Demonstration
        at com.mycompany.demonstration.Demonstration.main(Demonstration.j
ava:1)
Command execution failed.
```

Variable Scope

```
1   package com.mycompany.demonstration;
2
3   public class Demonstration {
4       public static void main(String args[]){
5           int a=0;
6           if (a==0){
7               int i = 5;
8           }
9           System.out.println(i);
10      }
11  }
```

Variable Scope

```
BUILD FAILURE
------------------------------------------------------------------------
Total time:  3.481 s
Finished at: 2022-11-11T08:42:45+08:00
------------------------------------------------------------------------
Failed to execute goal org.codehaus.mojo:exec-maven-plugin:3.0.0:exec (de
fault-cli) on project Demonstration: Command execution failed.: Process e
xited with an error: 1 (Exit value: 1) -> [Help 1]

To see the full stack trace of the errors, re-run Maven with the -e switc
h.
Re-run Maven using the -X switch to enable full debug logging.

For more information about the errors and possible solutions, please read
 the following articles:
[Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoExecutionEx
ception
```

5.1 Variable Scope

```
def sample(x):
    x[0] = 10
    x = [20, 30, 40]
    x[1] = 15
    print(x)


y = [5, 6, 7]
sample(y)
print(y)
```

Variable Scope

```
[20, 15, 40]
[10, 6, 7]
```

5.1 Variable Scope

```
y = [5, 6, 7]
sample(y)
print(y)


def sample(x):
    x[0] = 10
    x = [20, 30, 40]
    x[1] = 15
    print(x)
```

Variable Scope

```
Traceback (most recent call last):
  File "C:\Users\Dell\AppData\Local\Programs\
Python\Python310\demo.py", line 2, in <module
>
    sample(y)
NameError: name 'sample' is not defined
```

Variable Scope

```
Traceback (most recent call last):
  File "C:\Users\Dell\AppData\Local\Programs\
Python\Python310\demo.py", line 2, in <module
>
    sample(y)
NameError: name 'sample' is not defined
```

Variable Scope

```
1   package com.mycompany.demonstration;
2
3   public class Demonstration {
4
5       public static void main(String args[]){
6           hello();
7       }
8
9       static void hello(){
10          System.out.println("Hello Wrold");
11      }
12  }
```

Variable Scope

```
Building Demonstration 1.0-SNAPSHOT
--------------------------------[ jar ]---------------------------------

--- exec-maven-plugin:3.0.0:exec (default-cli) @ Demonstration ---
Hello Wrold
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
Total time:  1.493 s
Finished at: 2022-11-11T08:53:46+08:00
------------------------------------------------------------------------
```

Compare and Contrast

- Always remember sequence structures and best practices

Compare and Contrast

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

```
Hello World
```

Compare and Contrast

```
1  class Demo{
2      public static void main(String args[]){
3          System.out.println("Hello World");
4      }
5  }
```

```
Hello World
```

Functions that Reuse Code

```
def hello():
    print("Hello World")

hello()
```

Functions that Reuse Code

```
1   package com.mycompany.demonstration;
2
3   public class Demonstration {
4       static void hello(){
5           System.out.println("Hello World");
6       }
7
8       public static void main(String args[]){
9           hello();
10      }
11  }
```

6.1 Compare and Contrast

Functions that Return Value

```
def hello():
    return "Hello World"


print(hello())
```

Functions that Return Value

```
package com.mycompany.demonstration;

public class Demonstration {
    static String hello(){
        return "Hello World";
    }

    public static void main(String args[]){
        System.out.println(hello());
    }
}
```

Functions that Return Value

```
def hello():
    print("Hello World")
    return "Hello World"


print(hello())
```

Functions that Return Value

```
Hello World
Hello World
```

Functions that Return Value

```
1   package com.mycompany.demonstration;
2
3   public class Demonstration {
4       static String hello(){
5           System.out.println("Hello World");
6           return "Hello World";
7       }
8
9       public static void main(String args[]){
10          System.out.println(hello());
11      }
12  }
```

Functions that Return Value