Introduction to

Python

by W.S. Ooi

Python References

- https://www.programiz.com/python-programming
- https://www.python.org/
- Learn Python - Full Course for Beginners [Tutorial] https://www.youtube.com/watch?v=rfscVS0vtbw&t=5531s

First Python Script

Example 1:

- #!/usr/bin/env python
- import rospy
- print (“Hello”)

- Save the document as example1
- chmod +x /home/robot/example1
- /home/robot/example1

Explanation

- **1 #!/usr/bin/env python**
- **In order to run the python script, we need to tell the shell (Unix Interface) three things:**
  - That the file is a script
  - Which interpreter we want to execute the script
  - The path of said interpreter
- **The Unix shebang #! accomplishes (1.)**
- **The usr/bin/env python accomplishes (2.) and (3.)**
- **/usr/bin is a standard directory on Unix-like operating systems that contains most of the executable files**
- **Without this line, you have to type: Python example1**

Explanation

- ***chmod*** *refers to change mode, a command for changing file access permissions in a UNIX-based operating system*
- **chmod +x** on a file (your script) only means that you'll make it executable

Python - List

- List (or array) is data structure that contains ordered collection of elements
- Lists can contain numbers, strings, and any other data structures in an arbitrarily nested, heterogeneous fashion
- A list is surrounded by square brackets, and items are separated by comma
- List will be retaining the order in which they are created, unlike set. Hence indexing is completely possible
- Elements in a list is completely mutable, that is we can change them at any point in time
- List and string indexing works in the same fashion.

```
In [51]: marks = [95, 23, 50, 76, 65]
         type(marks)

Out[51]: list
```

```
In [54]: collections = ['A', 95, 'B', 23, 'C', 50, 'D', 76, 'E', 65]
         type(collections)

Out[54]: list
```

Python - List

```
In [55]: len(marks)

Out[55]: 5
```

```
In [58]: marks[0]

Out[58]: 95
```

```
In [80]: names = ['A', 'B', 'C', 'D', 'E']
         names[0:3]

Out[80]: ['A', 'B', 'C']
```

```
In [65]: for mark in marks:
             print (mark)

95
23
50
76
65
```

```
In [98]: marks.sort()
         marks

Out[98]: [23, 50, 65, 76, 95]
```

Python-Tuples

- Tuples is data structure that contains group of elements
- Tuples are similar to list, just that tuples are immutable i.e. the elements cannot be changed
- The syntax uses parenthesis instead of square brackets

```
In [4]: names_tuples = ('A', 'B', 'C', 'D', 'E')
        type(names_tuples)

Out[4]: tuple
```

- Elements in a tuples can be accessed similar to list. We can use the element's index to access them

```
In [8]: names_tuples = ('A', 'B', 'C', 'D', 'E')
        names_tuples[0]

Out[8]: 'A'
```

Python - Conditional Statements

- Conditional statements are used to test any assumptions or to compare values
- Conditional statements must be indented properly
- Simple conditional statement can have just the ***if*** part alone. When the condition specified inside the if part gets evaluated to true, then the statements within this block will get executed.
- You may also include an else block to handle the scenario, when the condition specified inside the if part gets evaluated to false.
- Nested conditional statements are also possible

```
In [38]: x = 1
         y = 2
         if x == y:
             print ('x and y are equal')
         else:
             print ('x and y are not equal')

         x and y are not equal
```

Python - Conditional Statements

In [46]:
```
x = 1
y = 2
if x > y:
    print ('x is greater than y')
elif x < y:
    print ('x is less than y')
else:
    print ('x is equal to y')
```

```
x is less than y
```

In [38]:
```
age = int(input(" Please Enter Your Age Here:  "))
if age < 18:
    print(" You are Minor ")
    print(" You are not Eligible to Work ")
else:
    if age >= 18 and age <= 60:
        print(" You are Eligible to Work ")
        print(" Please fill in your details and apply")
    else:
        print(" You are too old to work as per the Government rules")
        print(" Please Collect your pension!")
```

```
Please Enter Your Age Here:  7
You are Minor
You are not Eligible to Work
```

And/Or operator in conditional statements

In [56]:
```
x = 10
y = 20

if (x > 5) and (y > 5):
    print ('Both x and y are greater than 5')
elif (x > 5) or (y > 5):
    print ('Either x or y is greater than 5')
else:
    print ('Both x and y are less than 5')
```

```
Both x and y are greater than 5
```

Looping Statements

for loop: Repeats a set of statements over a group of values.

Syntax:

for variableName **in** groupOfValues:

    statements

- We indent the statements to be repeated with 4 white spaces.
- **variableName** gives a name to each value, so you can refer to it in the **statements**.
- **groupOfValues** can be a range of integers, specified with the range function.

```
In [2]: for x in range(1, 6):
            print(x, "squared is", x * x)
```

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

range()

- The range function specifies a range of integers:
- range(**start**, **stop**) - the integers between **start** (inclusive) and **stop** (exclusive)
- It can also accept a third value specifying the change between values.
- range(**start**, **stop, step**) - the integers between **start** (inclusive) and **stop** (exclusive) by **step**

```
In [4]: for x in range(5, 0, -1):
            print(x)
        print("Blastoff!")
```

```
5
4
3
2
1
Blastoff!
```

More Examples

In [5]:
```
for letter in 'Python':
    print('Current Letter :',letter)
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
```

In [7]:
```
fruits =['banana','apple','mango']
for fruit in fruits:
    print('Current fruit :',fruit)
```

```
Current fruit : banana
Current fruit : apple
Current fruit : mango
```

Looping Statements – While loop

while loop: Executes a group of statements as long as a condition is True. This is good for *indefinite loops* (repeat an unknown number of times.

Syntax:

while *condition*:

statements

```
In [10]: number = 1
         while number < 200:
             print(number)
             number = number * 2
```

```
1
2
4
8
16
32
64
128
```

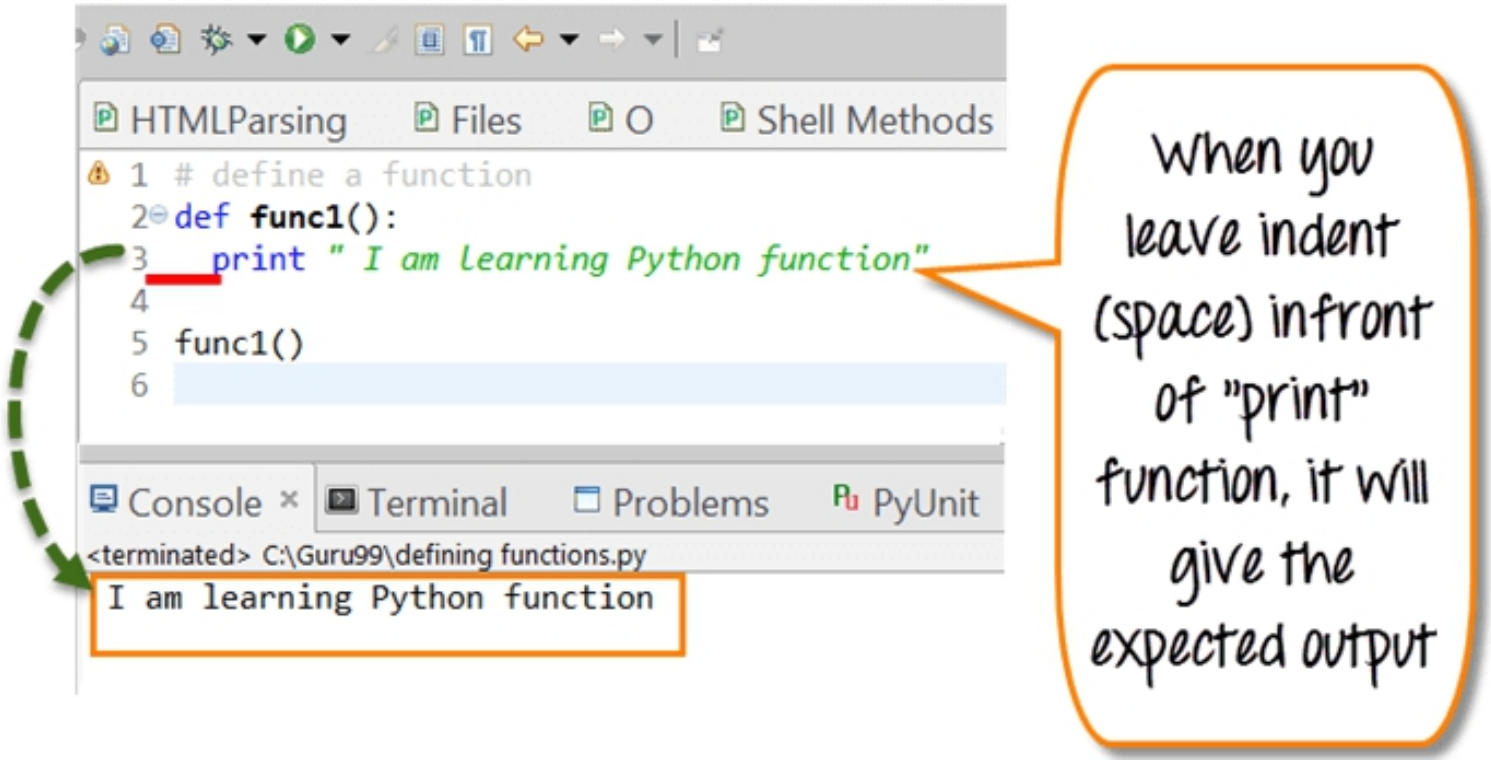
The else statement with loops

- Python supports an **else** statement associated with a loop statement.
- If the else statement is used with a for loop, the else statement is executed when the loop has exhausted iterating the list.
- If the else statement is used with a while loop, the else statement is executed when the condition becomes false.

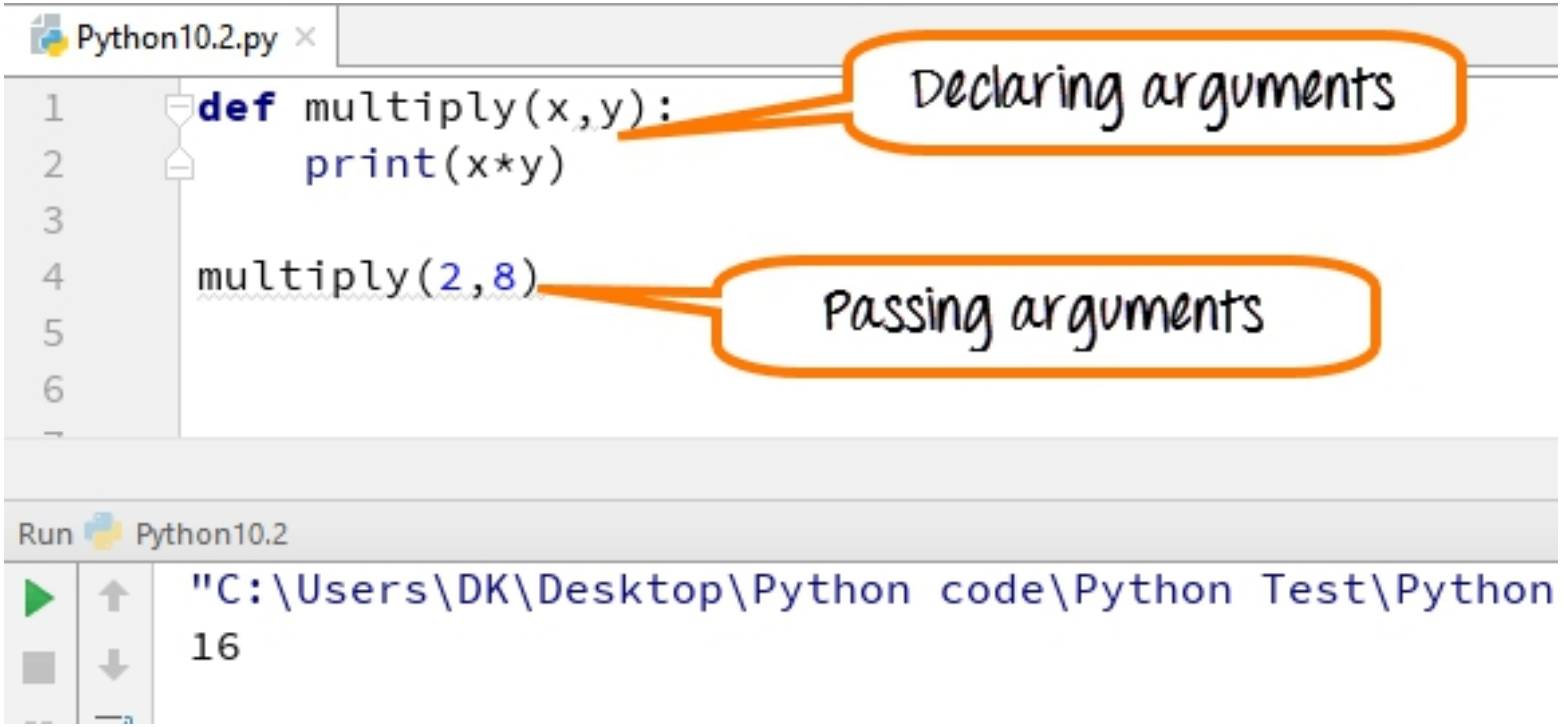
In [12]:
```
count =0
while count<5:
    print(count," is  less than 5")
    count=count+1
else:
    print(count," is not less than 5")
```

```
0  is  less than 5
1  is  less than 5
2  is  less than 5
3  is  less than 5
4  is  less than 5
5  is not less than 5
```

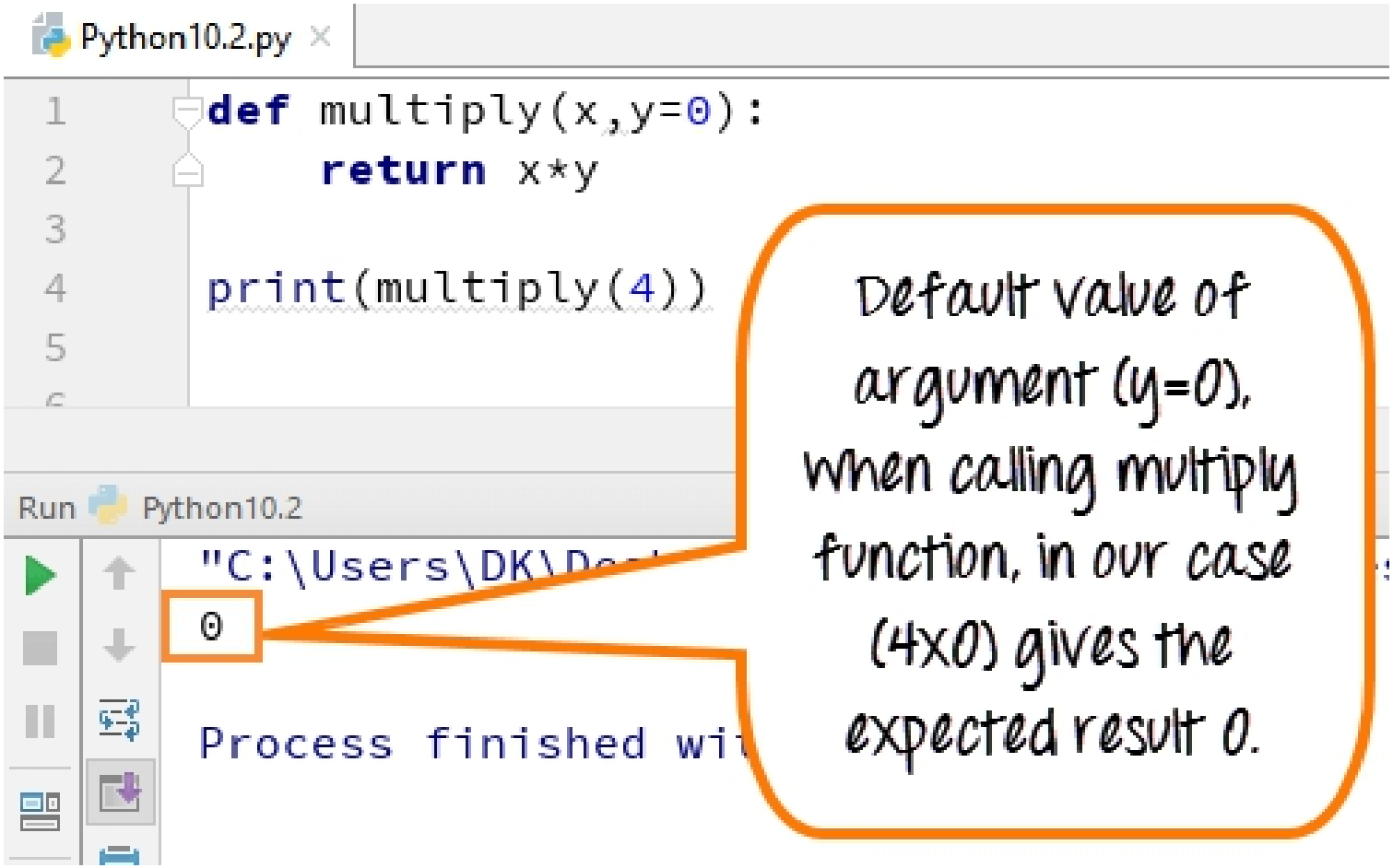
How to define a function?



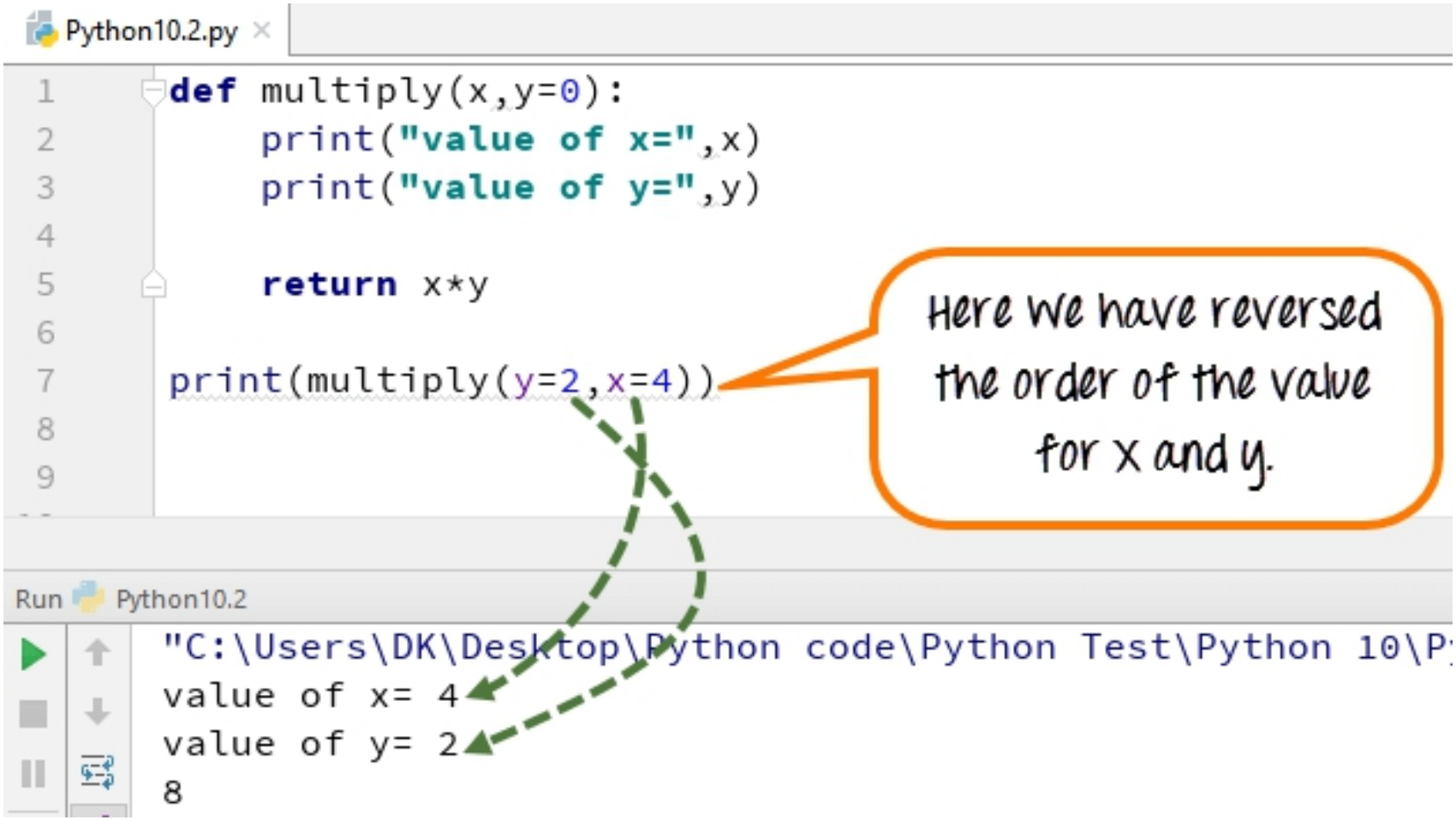
Function


Python10.2.py
1 def multiply(x,y):
2 print(x*y)
3
4 multiply(2,8)
5
6
Declaring arguments
Passing arguments
Run Python10.2
"C:\Users\DK\Desktop\Python code\Python Test\Python
16

Function


Python10.2.py
1 def multiply(x,y=0):
2 return x*y
3
4 print(multiply(4))
5
Run Python10.2
"C:\Users\DK\
0
Process finished wi
Default value of argument (y=0), when calling multiply function, in our case (4x0) gives the expected result 0.

Function


Python10.2.py
1 def multiply(x,y=0):
2 print("value of x=",x)
3 print("value of y=",y)
4
5 return x*y
6
7 print(multiply(y=2,x=4))
8
9
Here we have reversed the order of the value for x and y.
Run Python10.2
"C:\Users\DK\Desktop\Python code\Python Test\Python 10\P
value of x= 4
value of y= 2
8

PYTHON

Example 2:

- Create new document and type:
  - **1 #!/usr/bin/env python**
  - **2 import rospy**
  - **3 def myfunction():**
  - **4 print (“Hello”)**
  - **5 myfunction()**
- Save the document as myfunction1
- chmod +x /home/robot/myfunction1
- /home/computer/myfunction1

PYTHON

```
def myfunction(name):
        print(“Hello “ + name)
myfunction(“Manikam”)

def cube(num):
        return num*num*num
print(cube(3))
```

PYTHON

Example 3:

```
1 #!/usr/bin/env python
2 import rospy
3 def myfunction():
4       count=0
5       while (count<9):
6               print 'The count is:', count
7               count=count+1
8 print "Good bye!"
9 myfunction()
```

PYTHON

Example 4:

```
1 #!/usr/bin/env python
2 import rospy
3 def myfunction():
4     var=1
5     while var ==1:   #This constructs an infinite loop
6             num=raw_input("Enter a number :")
7             print "You entered:" , num
8 myfunction()
```

PYTHON

Example 4 modification:

```
1 #!/usr/bin/env python
2 import rospy
3 def myfunction():
4         var=1
5         while var ==1:   #This constructs an infinite loop
6                 num=raw_input("Enter a number :")
7                 print "You entered:" , num
8                     if int(num)<3:
9                             print "Enter less than 3"
10                                break
11        print "Good bye!"
12 myfunction()
```

Handling Exceptions

try:

Run this code

except:

Execute this code when there is an exception

else:

No exceptions? Run this code.

finally:

Always run this code.

```
def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
    else:
        print("result is", result)
    finally:
        print("executing finally clause")
```

```
divide(2,0)
```

```
division by zero!
executing finally clause
```

```
divide(5,2)
```

```
result is 2.5
executing finally clause
```

Handling Exceptions

In [1]:
```
def divide(x,y):

    result=x/y
    print(result)

divide(4,2)
```

```
2.0
```

In [2]:
```
def divide(x,y):

    result=x/y
    print(result)

divide(4,0)
```

```
---------------------------------------------------------------------------
ZeroDivisionError
```

In [3]:
```
def divide(x,y):

    try:
        result=x/y
        print(result)
    except:
        print("Error!")

divide(4,0)
```

```
Error!
```

In [4]:
```
def divide(x,y):

    try:
        result=x/y

    except:
        print("Error!")
    else:
        print(result)

divide(4,0)
```

```
Error!
```

In [5]:
```
def divide(x,y):

    try:
        result=x/y

    except:
        print("Error!")
    else:
        print(result)
    finally:
        print("Mission accomplished!")

divide(4,0)
```

```
Error!
Mission accomplished!
```

Python - Exception Handling

In [1]:
```
import math as m
def test():
    try:
        print(m.sqrt(-5))
        print(1)
    except:
        print("error")
        print(2)
    finally:
        print(3)
test()
```

```
error
2
3
```

PYTHON

Exercise 1:

- Write a Python program that will accept the base and height of a triangle and compute the area.

PYTHON

Exercise 1 answer:

- b = int(input("Input the base : "))
- h = int(input("Input the height : "))
- area = b*h/2
- print("area = ", area)

PYTHON

Exercise 2:

- Write a Python program to sort three integers

PYTHON

Exercise 2 answer:

- x = int(input("Input first number: "))
- y = int(input("Input second number: "))
- z = int(input("Input third number: "))
- a1 = min(x, y, z)
- a3 = max(x, y, z)
- a2 = (x + y + z) - a1 - a3
- print("Numbers in sorted order: ", a1, a2, a3)

PYTHON

Exercise 3:

- Write a Python program to check a triangle is equilateral, isosceles or scalene.
- Note :
  An equilateral triangle is a triangle in which all three sides are equal.
  A scalene triangle is a triangle that has three unequal sides.
  An isosceles triangle is a triangle with (at least) two equal sides.
  - Input lengths of the triangle sides:
  - x: 6
  - y: 8
  - z: 12
  - Scalene triangle

PYTHON

Exercise 3 answer:

```
print("Input lengths of the triangle sides: ")
x = int(input("x: "))
y = int(input("y: "))
z = int(input("z: "))
if x == y == z:
        print("Equilateral triangle")
elif x==y or y==z or z==x:
        print("isosceles triangle")
else:
        print("Scalene triangle")
```

PYTHON

Exercise 4:

BMI Calculator

Program in Python to compute the BMI of a person and display the risk associated with it by entering the height in 'm' and weight in' kg'. Refer the following table and code accordingly.

Note:

- To calculate the BMI apply the formula: BMI = weight(kg)/(height(m)*height(m)).
- Result must be adjusted to one decimal place.
- When the height or weight is entered as a negative number or zero, then display the message "Provide a valid input" and stop the program.

BMI	Risk
27.5 and above	High Risk
23 - 27.4	Moderate Risk
18.5 - 22.9	Low Risk
Below 18.5	Risk of nutritional deficiency diseases

PYTHON

Exercise 4:

Sample Input 1:
Enter the weight of the person(kg): 85
Enter the height of the person(m): 1.75
Sample Output 1:Your BMI is 27.8 (High Risk)

Sample Input 2:Enter the weight of the person(kg): 0
Enter the height of the person(m): 1.58
Sample Output 2:Provide a valid input

Sample Input 3:Enter the weight of the person(kg): 80
Enter the height of the person(m): -1
Sample Output 3:
Provide a valid input

PYTHON

Exercise 5:

Electricity Bill

Write a Python function, bill(), that asks the user enter the total units consumed. The other function, billcalc(), that receive the total units consumed and calculates the total electricity bill. The total bill will be passed back to the bill() function to print out the statement to the user. Refer the following tariff rate for calculating the total electricity bill. Note: If the unit consumed is entered as a negative number, then display " Invalid Unit" and stop the program.

TARIFF CATEGORY	UNIT	CURRENT RATE (1 JAN 2018)
Tariff A - Domestic Tariff		
For the first 200 kWh (1 - 200 kWh) per month	sen/kWh	21.80
For the next 100 kWh (201 - 300 kWh) per month	sen/kWh	33.40
For the next 300 kWh (301 - 600 kWh) per month	sen/kWh	51.60
For the next 300 kWh (601 - 900 kWh) per month	sen/kWh	54.60
For the next kWh (901 kWh onwards) per month	sen/kWh	57.10
The minimum monthly charge is RM3.00		

Introduction to

Robotic Operating System

(ROS)

by W.S. Ooi

What is ROS?

- The Robot Operating System (ROS) is a framework for writing robot software.
- It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

ROS Online Resources

- Main site: www.ros.org
- ROS Wiki: http://wiki.ros.org
- ROS Answers: http://answers.ros.org
- Some tutorials:
  - https://www.robotigniteacademy.com
  - http://edu.gaitech.hk/index.html
  - http://learn.turtlebot.com/
  - https://www.eng.yale.edu/grablab/roboticscourseware/courses.html

ROS Core Concepts

- Nodes
- Topics and Messages
- Services
- Actions
- ROS Master
- Parameters
- Packages
- Launch files

Nodes, Topics, and Messages

- http://wiki.ros.org/ROS/Tutorials

1. Core ROS Tutorials

1.1 Beginner Level

1. Installing and Configuring Your ROS Environment

   This tutorial walks you through installing ROS and setting up the ROS environment on your computer.

2. Navigating the ROS Filesystem

   This tutorial introduces ROS filesystem concepts, and covers using the roscd, rosls, and rospack commandline tools.

3. Creating a ROS Package

   This tutorial covers using roscreate-pkg or catkin to create a new package, and rospack to list package dependencies.

4. Building a ROS Package

   This tutorial covers the toolchain to build a package.

5. Understanding ROS Nodes

   This tutorial introduces ROS graph concepts and discusses the use of roscore, rosnode, and rosrun commandline tools.

6. Understanding ROS Topics

   This tutorial introduces ROS topics as well as using the rostopic and rqt_plot commandline tools.

7. Understanding ROS Services and Parameters

   This tutorial introduces ROS services, and parameters as well as using the rosservice and rosparam commandline tools.

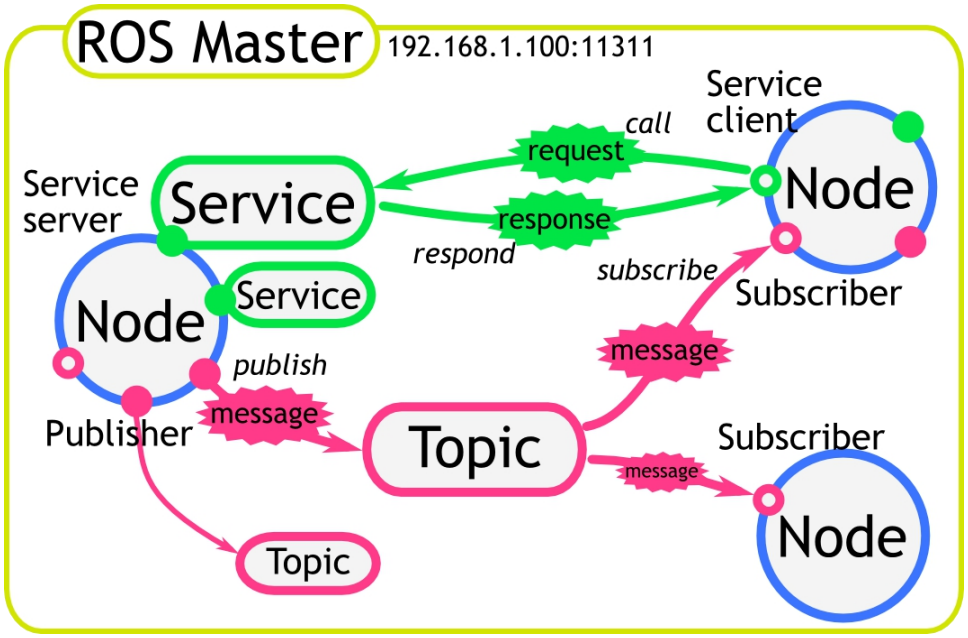
Nodes

Run ROS:
$ roscore

Run turtlesim:
$ rosrun turtlesim turtlesim_node

Play around with keyboard teleop:
$ rosrun turtlesim turtle_teleop_key

Display ROS computation graph:
$ rqt_graph



Nodes

- A *node* is a process that performs computation. A **node** is also an executable that uses **ROS** to communicate with other **nodes.**
- Nodes are combined together into a graph and communicate with one another using streaming topics, services, and the Parameter Server. These nodes are meant to operate at a fine-grained scale; a robot control system will usually comprise many nodes. For example, one node controls a laser range-finder, one Node controls the robot's wheel motors, one node performs localization, one node performs path planning, one node provides a graphical view of the system, and so on.
- The use of nodes in ROS provides several benefits to the overall system. There is additional *fault tolerance* as crashes are isolated to individual nodes. *Code complexity* is also reduced.

Turtlesim

Publish Once

- **rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]‘**

Publish at 1 Hz

- **rostopic pub -r 1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'**

Publish at 2 seconds

- **rostopic pub -r 0.5 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'**

- rosservice call /clear
- rosservice call /spawn 2 2 0.2 ""

Moving a turtle

- #!/usr/bin/env python
- import rospy
- import time
- from geometry_msgs.msg import Twist
- pub = rospy.Publisher('turtle1/cmd_vel', Twist, queue_size=10)
- rospy.init_node('mynode', anonymous=True)
- command = Twist()
- time.sleep(0.5)
- command.linear.x = 1.0
- command.angular.z = 0.0
- rospy.loginfo(command)
- pub.publish(command)

Moving a turtle

Save the script:

- Save the document as mymove
- chmod +x /home/robot/mymove

Run the turtlesim:

- robot@vm:~$ rosrun turtlesim turtlesim_node

Run the script to move the turtle:

- /home/robot/mymove

Explanation

- Import the Twist message from the geometry_msgs package. Twist data structure is used to represent velocity components

```
from geometry_msgs.msg import Twist
```

- rospy.init_node(), which initializes the ROS node for the process. In cases where you don't care about unique names for a particular node, you may wish to initialize the node with an *anonymous* name.

```
rospy.init_node('my node', anonymous=True)
```

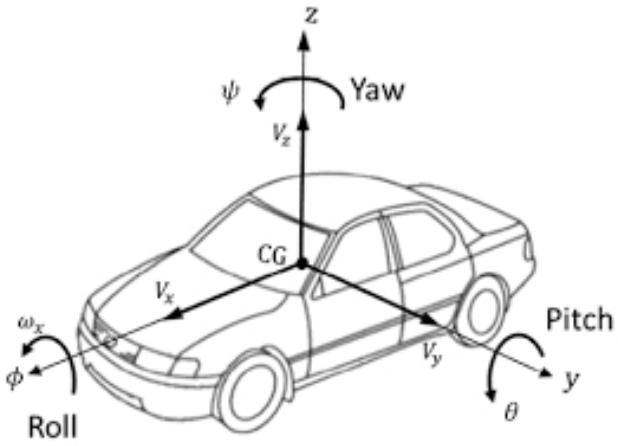
- The anonymous keyword argument is mainly used for nodes where you normally expect many of them to be running and don't care about their names. It adds a random number to the end of your node's name, to make it unique. Unique names are more important for nodes like drivers, where it is an error if more than one is running. If two nodes with the same name are detected on a ROS graph, the older node is shutdown.

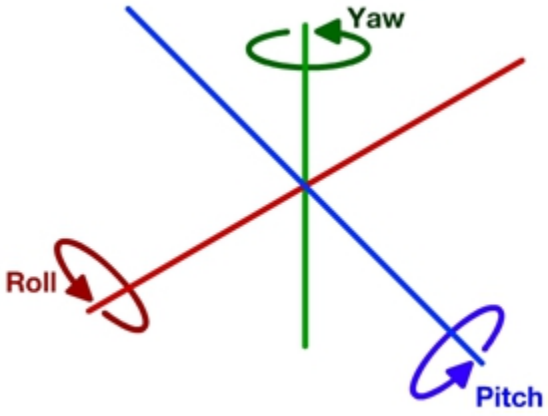
Explanation

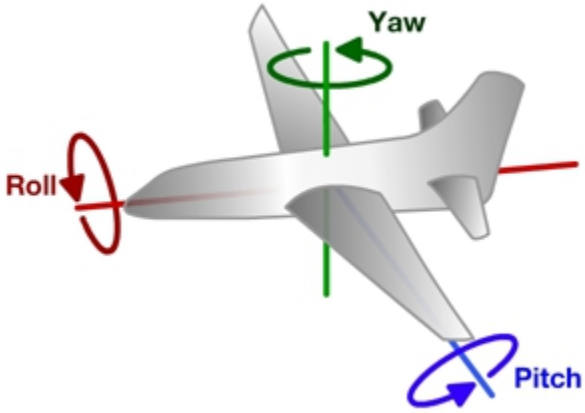
- pub = rospy.Publisher('topic_name', std_msgs.msg.String, queue_size=10)
- **mypub=rospy.Publisher(‘/turtle1/cmd_vel’, Twist, queue_size=10)**

- Choosing a good queue_size
- The 10 is the message queue size, that is, if you are publishing message faster then what roscpp can send, 10 messages will be saved in the queue to be sent. The larger the queue, the more delay in robot movement in case of buffering.
- Therefore in a real life example, you will want to have a smaller queue in the case of robot movement, where delay in movement commands are undesirable and even dangerous, but dropped messages are acceptable.
- In the case of sensors, it is recommended to use a larger queue, since delay is acceptable to ensure no data is lost.

Linear x and angular z?







Explanation

- time.sleep (second)

  - used to add delay in the execution of a program.

- rospy.loginfo(command)

  - display output to the console

Exercise:

1) Move you turtle follow a square shape
2) Spawn another turtle and move the second turtle follow a triangle shape after the first turtle completed a square shape.