DEPARTMENT OF
COMPUTER SYSTEMS ENGINEERING
MEHRAN UNIVERSITY OF ENGINEERING AND
TECHNOLOGY, JAMSHORO

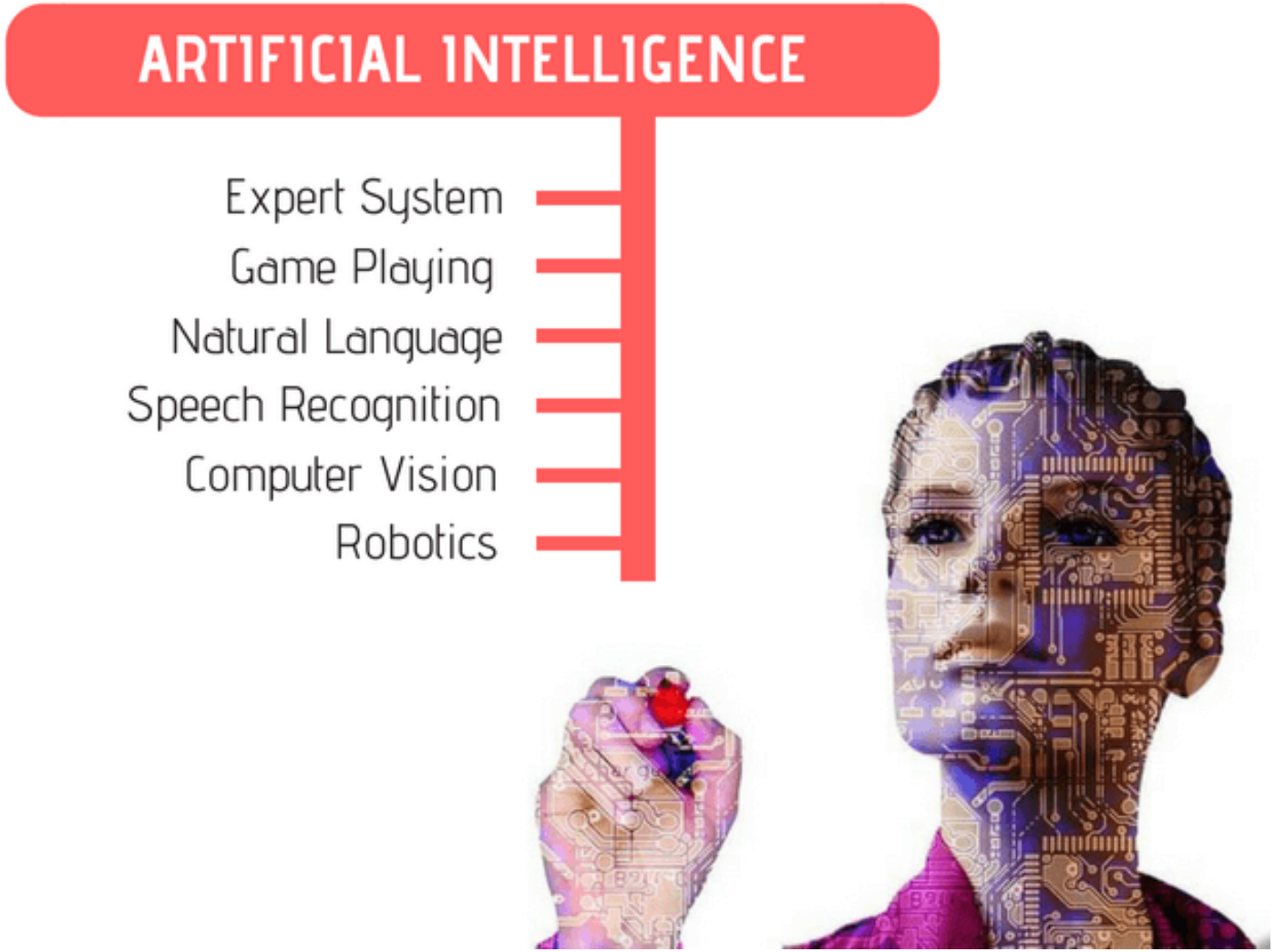
ARTIFICIAL INTELLIGENCE

PRACTICAL 01

OBJECT: ARTIFICIAL INTELLIGENCE WITH PYTHON

Outline

- AI practical applications
- Introduction to Python
- Getting started
- Variables
- Numbers
- Python Operators



Required Tools

- PC with windows
- Python 3.6.2

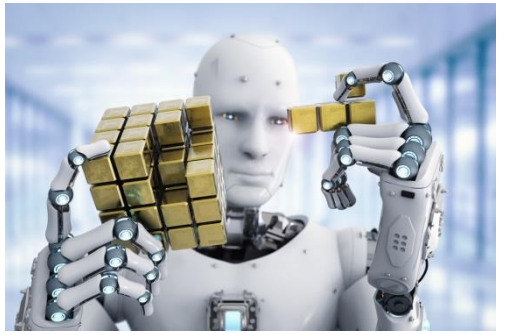
AI Applications

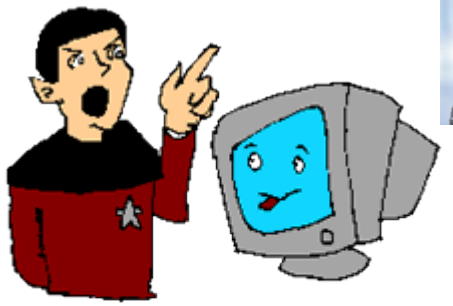
- **Gaming:** AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.

- **Natural Language Processing:** It is possible to interact with the computer that understands natural language spoken by humans.

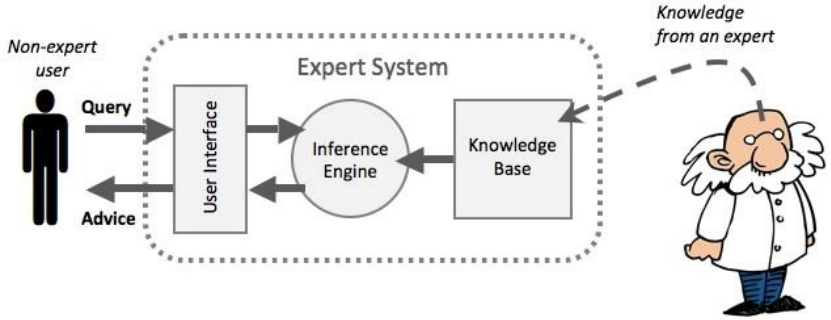
- **Expert Systems :** There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.

- **Vision Systems :** These systems understand, interpret, and comprehend visual input on the computer. For example,
  - ✓ A spying aero plane takes photographs, which are used to figure out spatial information or map of the areas.
  - ✓ Doctors use clinical expert system to diagnose the patient.
  - ✓ Police use computer software that can recognize the face of criminal with the stored portrait made by forensic artist.





Natural Language Processing



AI Applications

- **Speech Recognition:** Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talks to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.

- **Handwriting Recognition :**The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.

- **Intelligent Robots :**Robots are able to perform the tasks given by a human. They have sensors to detect physical data from the real world such as light, heat, temperature, movement, sound, bump, and pressure. They have efficient processors, multiple sensors and huge memory, to exhibit intelligence. In addition, they are capable of learning from their mistakes and they can adapt to the new environment.

INTRODUCTION TO PYTHON

What is Python?

- Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.
- It is used for:
- web development (server-side),
- software development,
- mathematics,
- system scripting.

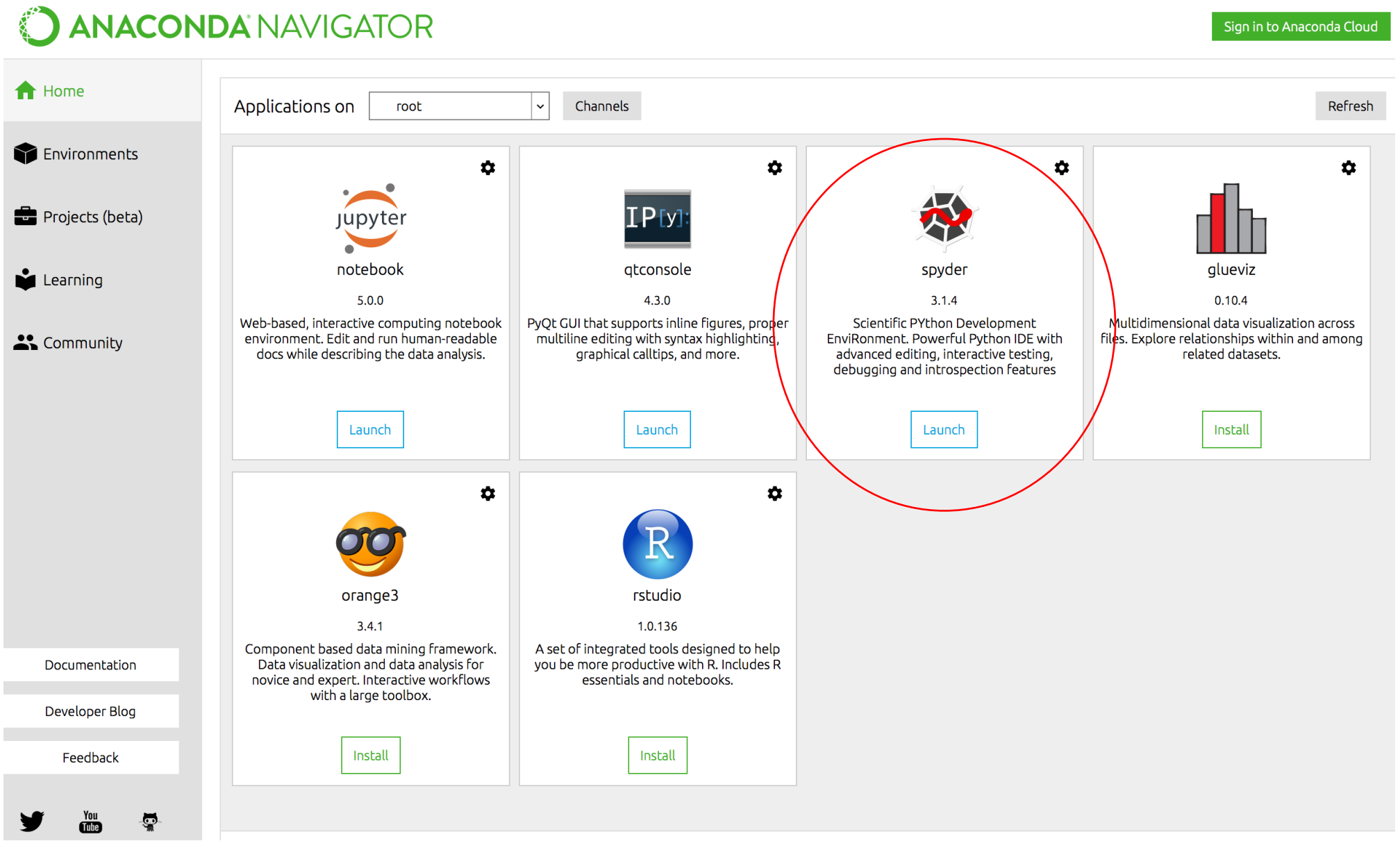
What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-orientated way or a functional way.

Getting Started



Getting Started

Let's write our first Python file, called **helloworld.py**, which can be done in any text editor.

```
print("Hello, World!")
```

TASK1: Print you Roll Number

TASK 2: write exit() in second line of code

Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

```
#This is a comment.
print("Hello, World!")
```

Docstrings

Python also has extended documentation capability, called docstrings.

Docstrings can be one line, or multiline.

Python uses triple quotes at the beginning and end of the docstring:

```
"""This is a
multiline docstring."""
print("Hello, World!")
```

PYTHON VARIABLES

Creating Variables

Unlike other programming languages, Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Variables do not need to be declared with any particular type and can even change type after they have been set.

Remember that variables are case-sensitive

```
x = 4 # x is of type int
X=5 #Value of x is changed
y= "Sally" # y is of type str
z= 3.14 #float type

print(x)
Print(y)
Prinr(z)
```

TASK3: Create a variable **country** with value as **pakistan**

Output Variables

The Python print statement is often used to output variables.

To combine both text and a variable, Python uses the + character:

```
x = "awesome"
print("Python is " + x)
---------------------------------------
x = "Python is "
y = "awesome"
z =  x + y  #For strings + works as Combination
print(z)
---------------------------------------
x = 5
y = 10
print(x + y) #For integers + works as operator
---------------------------------------
x = 5
y = "John"
print(x + y) # This will give an error
```

PYTHON NUMBERS

There are three numeric types in Python:

- int
- float
- Complex

To verify the type of any object in Python, use the type() function.

```
x = 1
y = 2.8
z = 5j

print(type(x))
print(type(y))
print(type(z))
```

How to get input?

```
print("Enter any number:")
x = input()
print("Number is:  ", x)
```

Python Casting

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

int() - constructs an integer number
float() - constructs a float number
str() - constructs a string

```
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
```

TASK4: Create two variables of type **int** & **float** and caste in into **string**

PYTHON OPERATORS

Operators are used to perform operations on variables and values.
Python divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Identity operators
- Membership operators
- Bitwise operators

TASK 5: Write a program to show whether the number is greater than, less than and equal to the other number.

TASK 5: Write a program to show add, sub, mul & div between two variables.

Arithmetic operators are used with numeric values to perform common mathematical operations:
(+, -, *, /, %(mod), **(exp) , //(floor div))

Comparison operators are used to compare two values: (==, !=, >, <, >=, <=)

Membership operators are used to test if a sequence is presented in an object: in, not in

PYTHON COLLECTIONS

There are four collection data types in the Python programming language:

- **List** is a collection which is ordered and changeable. Allows duplicate members.
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
- **Set** is a collection which is unordered and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

DICTIONARIES

Create and print a dictionary.

```
thisdict =
{
        "brand": "Ford",
        "model": "Mustang",
        "year": 1964
}
print(thisdict)
```

We can access the items of a dictionary by referring to its key name, inside square brackets:

```
x = thisdict["model"]
or
x = thisdict.get("model")
```

When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.

```
for x in thisdict: #print all key name
 print(x)
```

```
for x in thisdict: #print all values
 print(thisdict[x])
```

```
for x, y in thisdict.items(): #print both
print(x, y)
```

DICTIONARIES

Check if Key Exists

To determine if a specified key is present in a dictionary use the in keyword:

```
thisdict =
{
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

if "model" in thisdict:

print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"] = "red"
print(thisdict)
```

Python Conditions and If statements

Python supports the usual logical conditions from mathematics:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

These conditions can be used in several ways, most commonly in "if statements" and loops.
An "if statement" is written by using the if keyword.

If statement:

```
a = 33
b = 200
if b > a:
  print("b is greater than a") #space is mandatory
```

Elif : The elif keyword is pythons way of saying "if the previous conditions were not true, then try this condition".

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

Else : The else keyword catches anything which isn't caught by the preceding conditions.

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

Python Loops

Python has two primitive loop commands:

- ✓ while loops
- ✓ for loops

The while Loop

With the while loop we can execute a set of statements as long as a condition is true.

Example

Print i as long as i is less than 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

Note: remember to increment i, or else the loop will continue forever.

The break Statement

With the break statement we can stop the loop even if the while condition is true:

The continue Statement

With the continue statement we can stop the current iteration, and continue with the next:

Example

Exit the loop when i is 3:
Or continue to next interation if I is 3:

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break    # write continue here and show result
  i += 1
```

Python Loops

The FOR loop

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- This is less like the for keyword in other programming language, and works more like an iterator method as found in other object-orientated programming languages.
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

Print each fruit in a fruit list:

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Increment the sequence with 3 (default is 1):

```
for x in range(2, 30, 3):
  print(x)
```

Print all numbers from 0 to 5, and print a message when the loop has ended:

```
for x in range(6):
  print(x)
else:
  print("Finally finished!")
```

Nested Loops

A nested loop is a loop inside a loop.
The "inner loop" will be executed one time for each iteration of the "outer loop":

Print each adjective for every fruit:

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

PYTHON FUNCTIONS (User defined)

Creating a Function
In Python a function is defined using the def keyword:
Example

```
def my_function():
  print("Hello from a function")
```

call a function, use the function name followed by parenthesis:
my_function()

Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

PYTHON FUNCTIONS ()

Python abs()
returns absolute value of a number
Python bool()
Converts a Value to Boolean
Python chr()
Returns a Character (a string) from an Integer
Python complex()
Creates a Complex Number
Python dict()
Creates a Dictionary
Python divmod()
Returns a Tuple of Quotient and Remainder
Python float()
returns floating point number from number, string
Python format()
returns formatted representation of a value

Python help()
Invokes the built-in Help System
Python input()
reads and returns a line of string
Python len()
Returns Length of an Object
Python list() Function
creates list in Python
Python max()
returns largest element
Python min()
returns smallest element
Python next()
Retrieves Next Element from Iterator
Python pow()
returns x to the power of y
Python print()
Prints the Given Object

Python round()
rounds a floating point number to ndigits places.
Python set()
returns a Python set
Python str()
returns informal representation of an object
Python sum()
Add items of an Iterable
Python type()
Returns Type of an Object
Python globals()
returns dictionary of current global symbol table
Python hasattr()
returns whether object has named attribute
Python hash()
returns hash value of an object

EXERCISE

1. What is an expression?
2. What is a syntax error?
3. What is the result of this expression: “*” * 10.
4. What is a variable?
5. What are the primitive built-in types in Python?
6. When should we use “”” (tripe quotes) to define strings?
7. Assuming (name = “John Smith”)
    a. what does name[1] return
    b. What about name[-2]?
    c. What about name[1:-1]?
    d. How to get the length of name?

EXERCISE

8. Given (name = “john smith”), what will **name.title()** return?
9. What does name.**strip()** do?
10. What will **name.find(“Smith”)** return?
11. What will be the value of **name** after we call name.**replace(“j”, “k”)**?
12. How can we check to see if **name** contains **“John”**?
13. What are the 3 types of numbers in Python?
14. What is the difference between **10 / 3** and **10 // 3**?
15. What is the result of 10 ** 3?
16. Given (**x = 1**), what will be the value of after we run (**x += 2**)?
17. How can we round a number?
18. What is the result of **float(1)**?
19. What is the result of **10 == “10”**?
20. What is the result of **“bag” > “apple”**?
21. What is the result of **not(True or False)**?
22. What does **range(1, 10, 2)** return?

EXERCISE

23. Write a function that returns the maximum of two numbers.

24. Write a function called **check_num** that takes a number.
    - If the number is divisible by 3, it should return “Divisible by 3”.
    - If it is divisible by 5, it should return “Divisible by 5”.
    - If it is divisible by both 3 and 5, it should return “Divisible by both”.
    - Otherwise, it should return the same number.

25. Write a function called **showNumbers** that takes a parameter called **limit.** It should print all the numbers between 0 and limit.

26. Write a Python program which accepts the radius of a circle from the user and compute the area of all circle, triangle, square & rectangle.

EXERCISE

27. Write a function that returns the sum of multiples of 3 and 5 between 0 and **limit** (parameter). For example, if limit is 20, it should return the sum of 3, 5, 6, 9, 10, 12, 15, 18, 20.

28. Write a function called **show_stars(rows).** If **rows** is 5, it should print the following:

```
*
**
***
****
*****
```

29. Write a function that prints all the prime numbers between 0 and **limit** where limit is a parameter.

30. Write a Python program to display the first and last colors from the following list.

```
color_list = ["Red","Green","White" ,"Black"]
```

EXERCISE

31. Write a Python program which accepts the user's first and last name and print them in reverse order with a space between them.

32. Write a Python program to find whether a given number (accept from the user) is even or odd, print out an appropriate message to the user.

33. Write a Python program that accepts an integer (n) and computes the value of n+nn+nnn.

34. Write a Python program to test whether a passed letter is a vowel or not.

35. Write a Python program to sum of three given integers. However, if two values are equal sum will be zero.

36. Write a Python program that will return true if the two given integer values are equal or their sum or difference is 5.

37. Write a Python program to solve (x + y) ^(z).
    Test Data : x = 4, y = 3, z=2
    Expected Output : (4 + 3) ^ 2) = 49