PYTHON PROGRAMMING

UNIT 1
Introduction to Python

Unit 1 Outline I

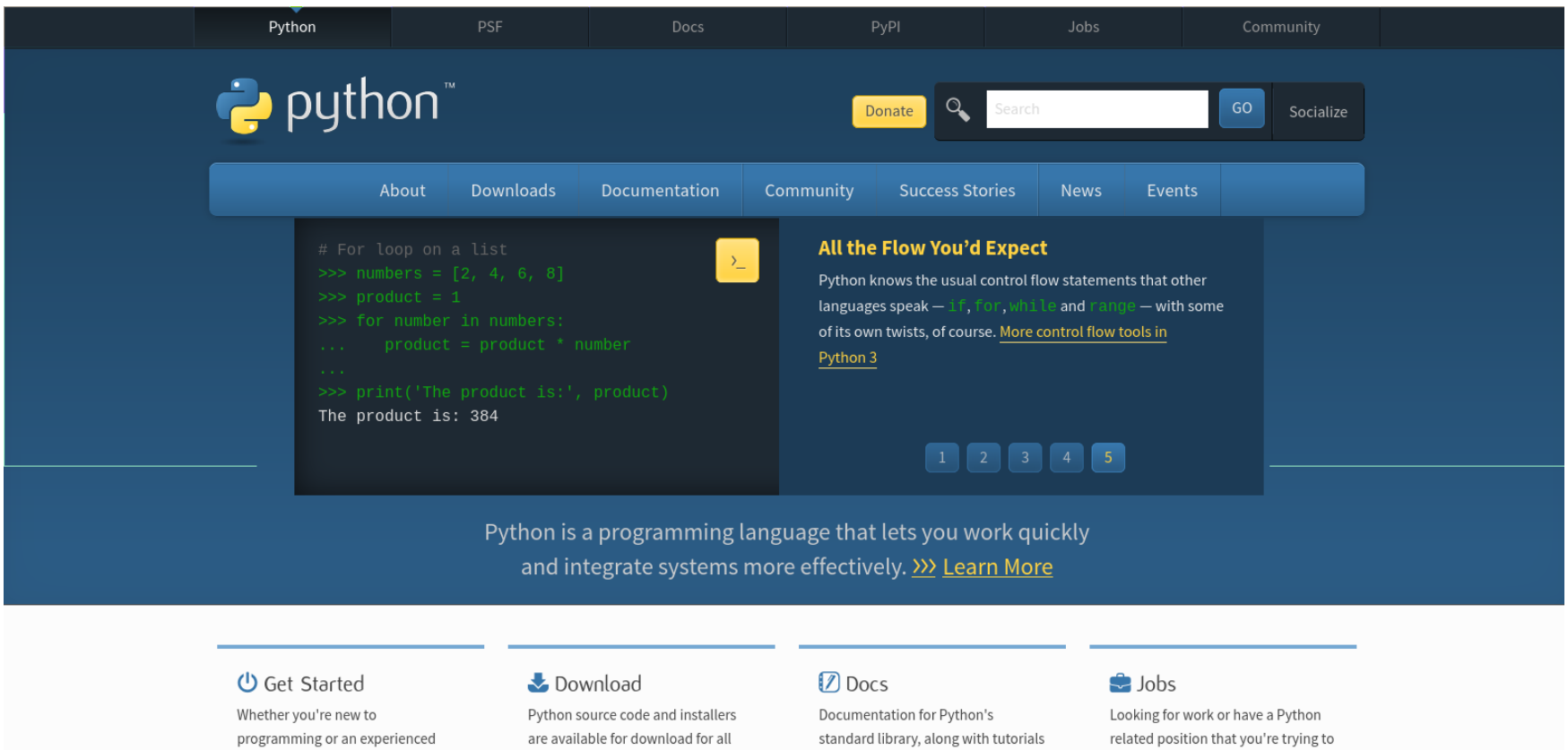
Unit 1 Outline II

- Comparison Operators
- Membership Operators
- Identity Operators
- Bitwise operators
- Precedence and Associativity

4 Input/Output Functions

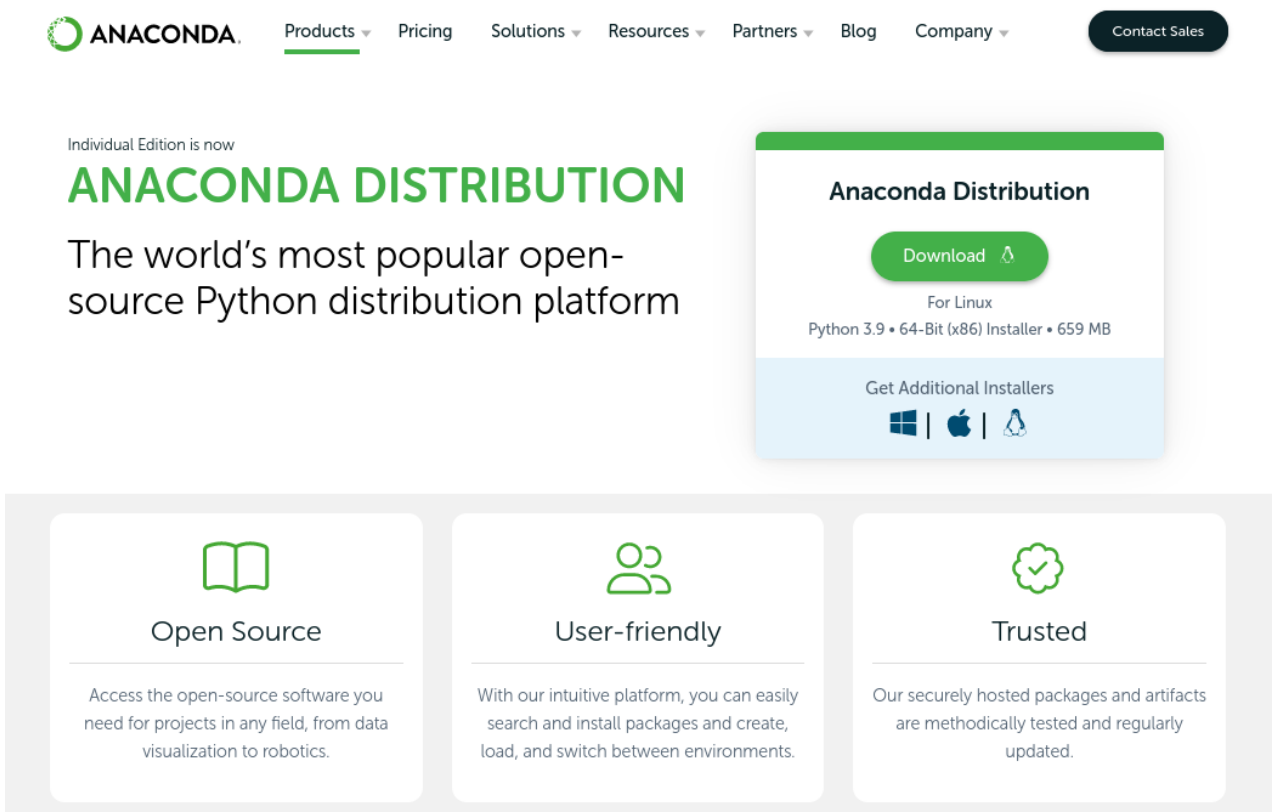
Python Installation

- Python is an open source programming language.
- The source code for python along with documentation and tutorials can be found at `https://www.python.org/`



Python Installation

- Anaconda distribution provides an easy way to search and install thousands of python packages
- Anaconda distribution can be obtained at `https://www.anaconda.com/products/distribution`



Python Installation

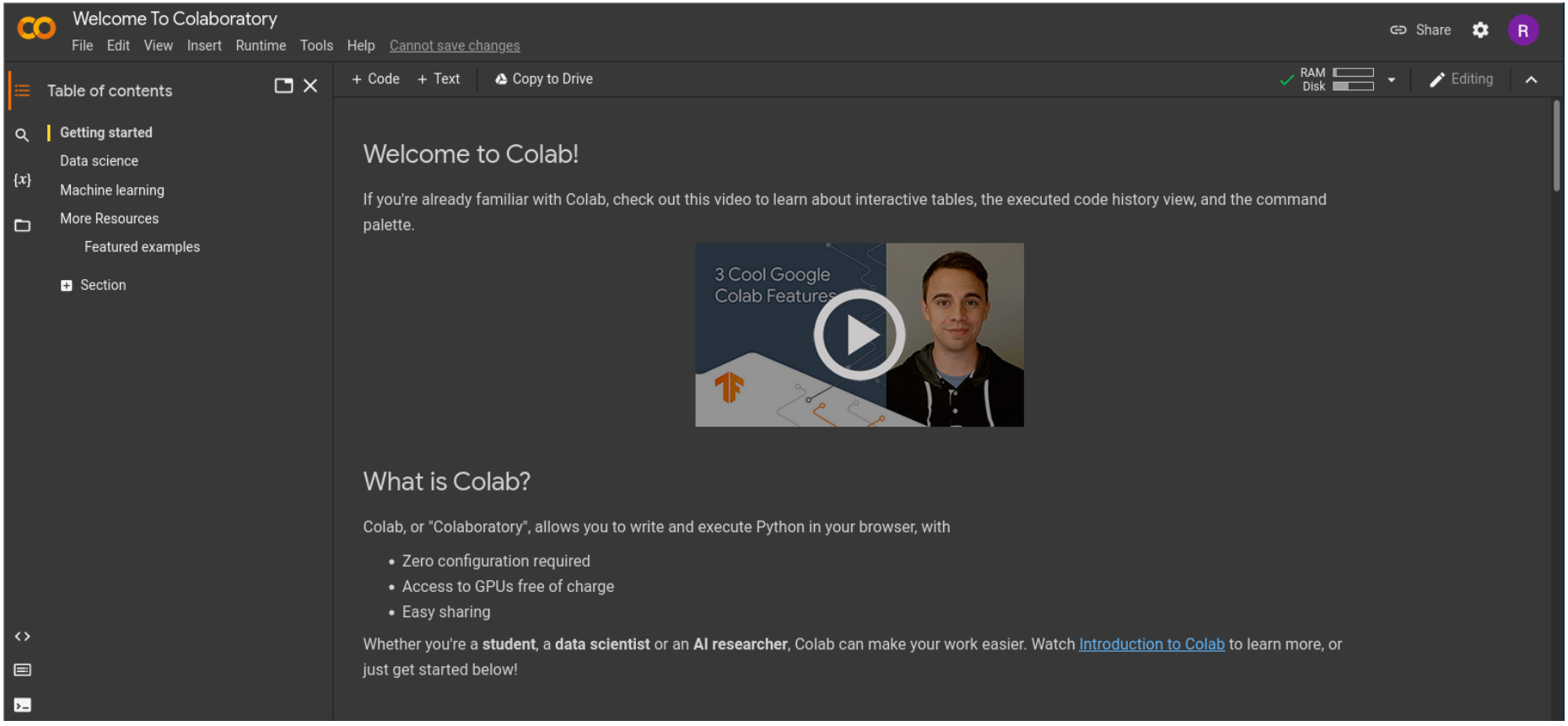
- A Jupyter notebook is an application that can run Python code, display plots, show equations and contain formatted text
- Jupyter notebooks are a great tool for problem solvers to write, run, document and share Python code with others
- The Python code in a Jupyter notebook is the same type of Python code found in a .py file
- The Anaconda distribution of Python comes with Jupyter notebook included and no further installation steps are necessary

Installation Key

The Anaconda distribution of Python comes with Jupyter notebook included and no further installation steps are necessary.

Python Installation

- Google Colaboratory allows to run python program without requiring any python installation at `https://colab.research.google.com/`



Python Programming

- Two types of programming languages
  - Low level languages (Machine language or Assembly language)
  - High level languages (Python, C, C++, Perl, Java)
- The program written in high level languages are converted into low language (machine code) through **compilers** or **interpreters**

Figure 1.1: An interpreter processes the program a little at a time, alternately reading lines and performing computations.

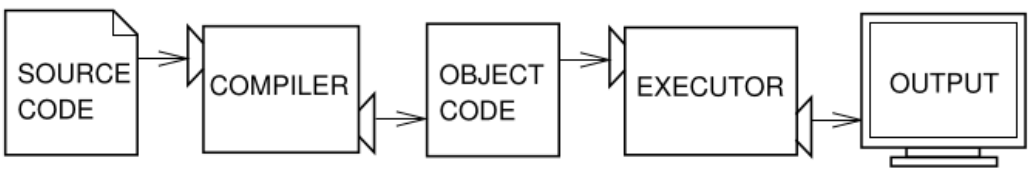
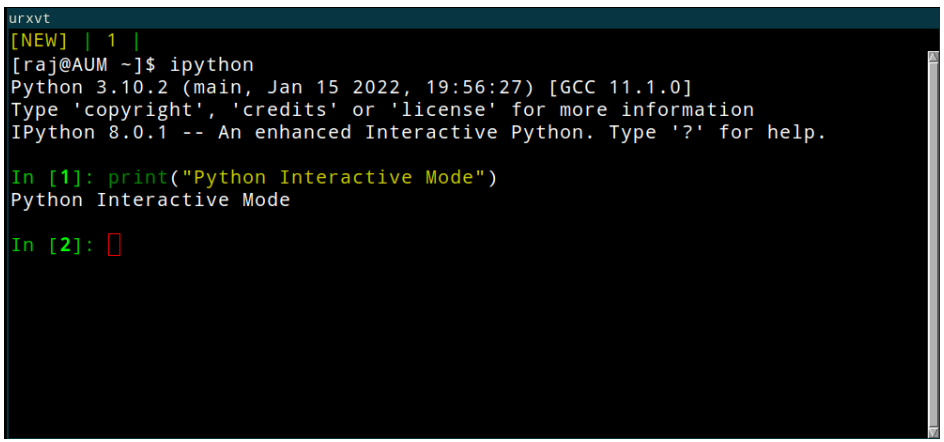


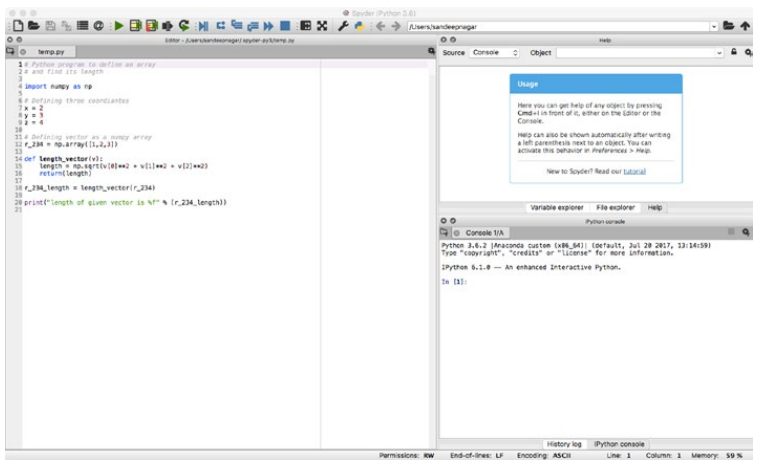
Figure 1.2: A compiler translates source code into object code, which is run by a hardware executor.

Python Programming

- Python is an interpreted language where instructions are executed step-by-step by an interpreter
- There are two ways to use the interpreter:
  - Interactive mode
  - Script mode



Interactive Mode



Script Mode

Python Programming

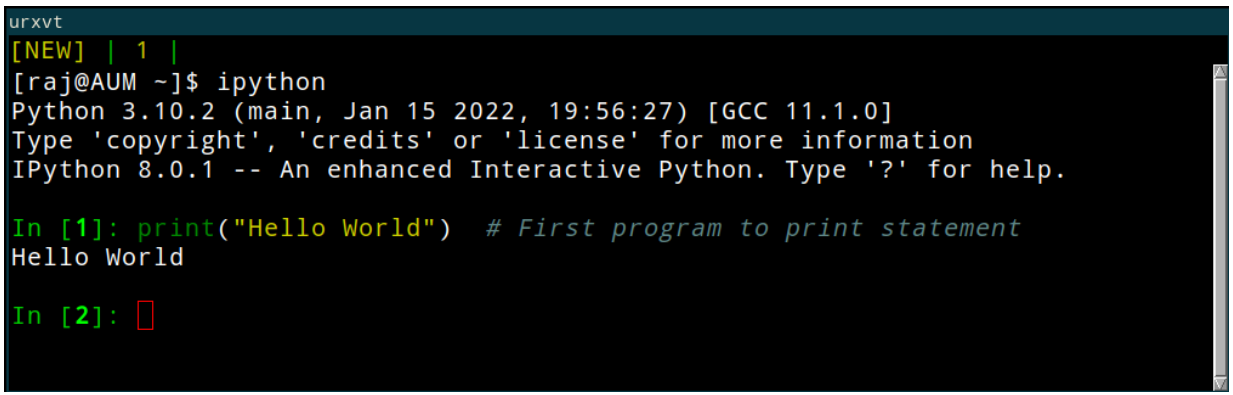
- Python is very popular programming languages providing several packages for performing Numerical, Scientific programming including Artificial Intelligence, Machine Learning and Data Analysis
- Being open source in nature, python allowed people to customize
- Python enables individual to learn program easily, to read and write program in simple and readable manner
- Python is a modular language (Add-on packages)
- Each module is designed to handle a specific task

Features

Python is a multipurpose, portable, object-oriented, high-level programming language that enables an interactive environment to code in a minimalistic way

Python Programming

- A Python program, sometimes called a script, is a sequence of definitions and commands.
- A command, often called a statement, instructs the interpreter to do something.



Python Program

Python Programming

- We will use Integrated development environment (IDEs) for Python coding
- Some of them being PyCharm, Jupyter Notebook and Google Colab
- Jupyter Notebook belongs to Anaconda

Variables

- A **value** in a program could be letter or numbers such as 1, 2.3, 'a', 'abc'
- To store or to assign value temporarily during computation **variables** are use
- Variables point to a particular value at a memory location via its address

```
urxvt
[NEW] | 1 |
[raj@AUM ~]$ ipython
Python 3.10.2 (main, Jan 15 2022, 19:56:27) [GCC 11.1.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.0.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: a = 1  # assigning value 1 to variable 'a'

In [2]: print(a)  # print value associated with variable 'a'
1

In [3]: a = 2.8  # Reassigning value to variable 'a'

In [4]: print(a)  # print value associated with variable 'a'
2.8

In [5]: b = "a"  # assigning value 'a' to variable 'b'

In [6]: print(b)  # print value associated with variable 'b'
a

In [7]: print(a)  # print value associated with variable 'a'
2.8
In [8]:
```

Data Types

- Data can be numbers, characters, strings (a group of characters)
- Python contains several built-in data types and user can define new data types as well
- Different data types occupy different amount of memory
  - Numeric (`int`, `float`, `complex`)
  - Strings
  - Lists
  - Tuples
  - Dictionary
  - Set and Frozen sets
  - Logical (`AND`, `OR`, `TRUE`, `FALSE`)
  - `None` (Null Objects)
- In python, type of data types can be obtained using `type()` function

Data types : Numeric

- `int` are positive and negative whole numbers (comments are written followed by # in python)

```
>>> # 'a' and 'b' variables int type
>>> a = 10000
>>> b = -10000
>>> type(a)        # to find data type of 'a'
<class int'>
```

- Real numbers (+/-) with decimal points and fractional part is represented in python as `float` data types

```
>>> # float data types
>>> a = -1.0
>>> b = 0.123
>>> c = -12.34
>>> d = 1.23e-4     # using scientific notation
>>> e = -1.23e4
>>> type(e)         # to find data type of 'e'
<class float'>
>>> f = int(c)      # type conversion from float to int
>>> print(f)
-12
>>> type(f)
<class int'>
```

Data types : Numeric

- Complex numbers are extensively used in science and engineering studies
- A complex number can also be defined using the built-in function `complex()`

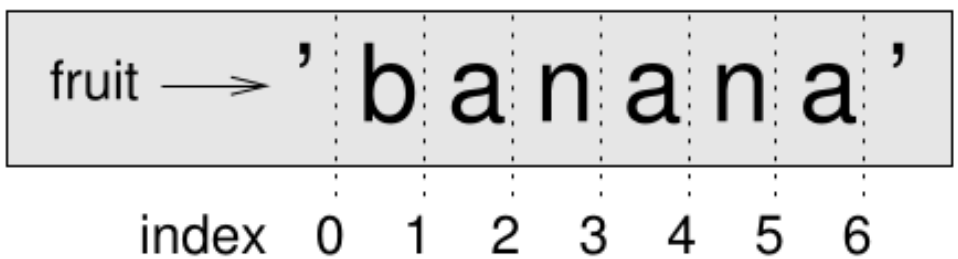
```
>>> a = 1 + 2.3j
>>> print(a)
(1+2.3j)
>>> type(a)
<class complex'>
>>> a = complex(1,2.3)
>>> print(a.real, a.imag)
1.0 2.3
>>> a.conjugate()
(1 - 2.3j)
>>> b = 4.5 + 6j
>>> c = a + b
>>> print(c)
(5.5+8.3j)
>>> d = a*b
>>> print(d)
(-9.299999999999999+16.35j)
```

Data types : Strings

- A string is a sequence of characters
- Lowercase and uppercase characters have different encoding and hence strings are case-sensitive

```
>>> a = 'a'
>>> b = 'abc'
>>> name = 'python'
>>> type(name)
<class str'>
>>> print(name)
python
```

- The characters of a string can be accessed through indexing
- In python, all indexing is zero-based for example, typing `name[0]` into the interpreter will cause it to display the string character `'b'`



Data types : Strings

- The extraction of substrings (**slice**) from a string is called **slicing**

```
urxvt
[NEW] | 1 | 2 |
Python 3.10.5 (main, Jun  6 2022, 18:49:26) [GCC 12.1.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.4.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: name = "Python Programming"

In [2]: type(name)
Out[2]: str

In [3]: name[0]
Out[3]: 'P'

In [4]: name[-1]
Out[4]: 'g'

In [5]: name[0:6]
Out[5]: 'Python'

In [6]: name[:6]
Out[6]: 'Python'

In [7]: name[-11:]
Out[7]: 'Programming'

In [8]: language = name[0:6]   # Assigning 'python' from name variable

In [9]: print(language)
Python

In [10]:
```

Data types : Strings

- In python several operations can be performed on strings using built-in functions and overloaded operators

```
>>> s1 = "red"
>>> s2 = "apple"
>>> len(s1)     # find length of string 's1'
3
# concatenate two strings and assign it to 's3'
>>> s3 = s1 + ' ' + s2
>>> print(s3)
red apple
# applying different methods to strings
>>> s2.upper()
'APPLE'
>>> s2.find('pl')
2
```

Data types : Lists

- Similar to strings, a list is an ordered set of objects, irrespective of its data type
- The values in a list are called elements or sometimes items.
- The simplest way to create the list is to enclose the elements in square brackets ([and])

```
>>> a = [1, 2.3, 'string']
>>> print(a)
[1,2.3, 'string']
>>> type(a)
<class list'>
>>> b = list([1, 2.3, 'string'])
>>> type(b)
<class list'>
>>> c = []      # empty list
```

Data types : Lists

- The elements of list can be accessed through indexing similar to the strings

```
>>> a = ['string', 1.2, 3]
>>> a[0]
'string'
>>> print(a[1])
1.2
>>> a[-1]
3
>>> b = a[1]
>>> b
1.2
>>> type(b)
<class float'>
>>> c = a[0]
>>> print(c)
string
>>> type(c)
<class str'>
```

Data types : Lists

- The elements of list can assigned to another list variable

```
>>> a = ['a','b','c','d','e']
>>> b = a[0:3]   # list slice
>>> print(b)
['a','b','c']
```

- Unlike strings, lists are mutable

```
>>> a = ['apple', 'orange', 'banana']
>>> a[1]
'orange'
>>> a[1] = 'watermelon'
>>> print(a)
['apple', 'watermelon', 'banana']
```

Data types : Tuples

- Tuples are types of sequences similar to lists defined using parentheses ( ) instead of brackets [ ]
- The important difference between lists and tuples is that tuples are immutable meaning that their elements, once defined, cannot be altered
- Tuple is a comma-separated list of values

```
>>> a = 1, 2, 3  # comma separated values, comma must for tuple
>>> print(a)
(1,2,3)
>>> type(a)
<class tuple'>
>>> b = ('a','b','c','d')
>>> c = ('a')   # c is not tuple
>>> type(c)
<class str'>
>>> d = tuple("python")
>>> print(d)
('p','y','t','h','o','n')
```

Data types : Tuples

- **Slicing** in tuple works similar to list variable

```
>>> a = (1,2,3)
>>> a[1]
2
>>> a[2] = 3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment

# values can be assigned in tuple to multiple variables
>>> (a,b) =  (1,2)
>>> print(a)
1
>>> a,b,c = 1,2,3
>>> a = (1,2,3)
>>> b = (4,5)
>>> c = a + b
>>> print(c)
(1,2,3,4,5)
>>> d = a*2
>>> print(d)
(1,2,1,2)
```

Data types : Dictionary

- A dictionary organizes information by association, not position
- In Python, a dictionary associates a set of keys with data values
- The entire sequence of entries is enclosed in curly braces ({ and })
- A colon (:) separates a key and its value

```
>>> data = {'name':'student name','inst':'AU','dept':'CSE/ICT'}
>>> type(data)
<class dict'>
>>> print(data)
{'name': 'student name', 'inst': 'AU', 'dept': 'CSE/ICT'}
>>> data['name']
'student name'
>>> data['inst']
AU
>>> data['dept'] = 'CSE-ICT'       # modifying value
>>> data = {}                      # empty dictionary
>>> data['name'] = 'student name' # adding key-value pair
>>> data['inst'] = 'AU'
>>> data['dept'] = 'CSE-ICT'
>>> print(data)
{'name': 'student name', 'inst': 'AU', 'dept': 'CSE-ICT'}
```

Data types : Sets and Frozensets

- Sets and Frozensets are an unordered collection of objects
- Unlike sequence objects, such as list and tuple, where elements are ordered, sets do not have such requirements
- However, sets do not permit duplicity in the occurrence of an element
- A set is defined using the `set()` function, which is supplied a list as its input argument or it can be defined using curly brackets { and }

```
>>> a = set([1,1,2,3,4,2,4,'a'])
>>> print(a)
{1,2,3,4,'a'}
>>> type(a)
<class set'>
>>> b = {'d','b','c','a','a','c','e', 1,2,1}
>>> print(b)
{1,2,'a','c','e','b','d'}
```

Data types : Sets and Frozensets

- The set type is mutable, meaning it can be changed, and the frozenset type is immutable, meaning it can't be changed.

```
>>> a = set([1,1,2,3,4,2,4, 'a'])
>>> print(a)
{1,2,3,4,'a'}
>>> a.add{'c'}
>>> print(a)
{1,2,3,4,'a','c'}
>>> b = {'d','b','c','a','a','c','e', 1,2,1}
>>> print(b)
{1,2,'a','c','e','b','d'}
>>> a.intersection(b)
{1,2,'a','c'}
>>> a.union(b)
{1,2,3,4,'c','b','d','a','e'}
>>> c = frozenset([8,5,7,2,3,6])
>>> c = frozenset({8,5,7,2,3,6})
>>> print(c)
frozenset({[8,5,7,2,3,6})
```

Data types : Logical

- This type of data stores boolean values `True` or `False` and can be operated by boolean operators such as `AND` and `OR`

```
>>> a = True
>>> print(a)
True
>>> type(a)
<class bool'>
>>> b = False
>>> a and b
False
>>> a or b
True
>>> c = 123
>>> d = 45
>>> c > d
True
```

Data types : Null

- None is a null object
- It refers to nonfunctionality means no behavior for the object with which it is associated
- When it is issued at the python prompt, nothing happens

```
>>> None
>>> a = None
>>> a
>>> print(a)
None
>>> type(a)
<class NoneType'>
```

Python Operators

- To perform various types of mathematical and logical operations, Several types of operators are used in python
- Operators works similar to mathematical functions
- Various operators can be combined to perform an arithmetic operations
- Depending upon the data types, operators functionality may vary
- For example '+' operator on `int` and `float` performs summation whereas on `str` data type, it performs concatenation operation
- Some operators are defined for certain data types only for example division, modulus and exponential operators are not defined for `str` data type

Assignment Operators

- The symbol = assigns the value on the right-hand side to the variable name on the left-hand side (links memory location)
- The assignment operator = is not the same as 'equal to' in mathematics (comparison operator == is used to equate two values or two variables)

Operator	Example
=	v = a+b
+=	v +=a ⇒ v = v + a
-=	v -=a ⇒ v = v - a
/=	v /=a ⇒ v = v / a
//=	v //=a ⇒ v = v // a
*=	v *=a ⇒ v = v * a
**=	v **=a ⇒ v = v ** a
%=	v %=a ⇒ v = v % a

Assignment Operators

Assignment Operators

```
>>> a = 123
>>> a = b = c = 123    # multiple assignment
>>> a += 50            # summation equivalent to a = a + 50
>>> print(a)
173
>>> a -= 50
>>> print(a)
123
>>> a *= 2
>>> print(a)
246
>>> a /= 2
>>> print(a)
123
>>> a **= 2
>>> print(a)
15129
```

Arithmetic Operators

- Arithmetic operators (+, -, *, /,. . .) in python operates similar to the operators in mathematics
- a^b is written as a**b in python

Operation	Description	Example
Addition	Adds value on either side of the operator	x+y
Subtraction	Subtracts right operand from left operand	x-y
Multiplication	Multiples values on either side	x*y
Division	Divides left operand with right one	x/y
Modulus	Returns remainder in division	x%y
Exponent	Returns powers to a operand	x**y

Arithmetic Operators

Arithmetic Operators

```
>>> a = 2.3 + 4.5
>>> print(a)
6.8
>>> b = a - 1.2
>>> print(b)
5.6
>>> a = b*2
>>> print(a)
11.2
>>> a = b**2
>>> print(a)
31.359999999999996
>>> a = a/b
>>> print(a)
5.6
>>> print(a % 1.3)
0.3999999999999947
```

Arithmetic Operators

- When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence (PEMDAS)
- Parentheses have the highest precedence for example expression $2+3*2+3$ evaluated as 11 whereas $(2+3)*(2+3)$ evaluated as 25
- Exponentiation has the next highest precedence so $2**1+1$ is 3, not 4, and $3*1**3$ is 3, not 27
- Multiplication and Division have the same precedence, which is higher than Addition and Subtraction, which also have the same precedence So $2*3-1$ is 5, not 4, and $6+4/2$ is 8, not 5
- Operators with the same precedence are evaluated from left to right (except exponentiation) so in $1/2*pi$ expression division happens first and then multiplication instead of multiplication of $2*pi$ we expect

Arithmetic Operators

EXPRESSION	EVALUATION	VALUE
`5 + 3 * 2`	`5 + 6`	`11`
`(5 + 3) * 2`	`8 * 2`	`16`
`6 % 2`	`0`	`0`
`2 * 3 ** 2`	`2 * 9`	`18`
`-3 ** 2`	`-(3 ** 2)`	`-9`
`(3) ** 2`	`9`	`9`
`2 ** 3 ** 2`	`2 ** 9`	`512`
`(2 ** 3) ** 2`	`8 ** 2`	`64`
`45 / 0`	`Error: cannot divide by 0`	
`45 % 0`	`Error: cannot divide by 0`	

Rules of precedence

Comparison Operators

- To compare objects, various logical or comparison operators are used

Operator Symbol	Operator Meaning	Example
==	equal to	1==1 is True, 1==2 is False
!=	not equal to	1!=1 is False, 1==2 is True
<	less than	1<2 is True, 2<1 is False
>	greater than	1>2 is False, 2>1 is True
<=	less than equal to	1<=1 is True, 1<=2 is True
>=	greater than equal to	1>=1 is True, 1>=2 is False

Comparison Operators

Comparison Operators

```
>>> a,b = 1,2
>>> print(a == 1)
True
>>> print(b != 2)
False
>>> print(a < 0)
False
>>> print((a > 0) and (b > 0))
True
>>> print(a >= 2)
False
>>> print(b <= 2)
True
```

Membership Operators

- Membership operator `in` checks if values of variables is a member of a specified sequences such as strings, lists, tuples, dictionary, sets
- If the member is found, it returns the boolean value `True` otherwise returns `False`
- The operator `in` is used extensively in checking conditions for loops

```
>>> 'python' in 'python programming'
True
>>> 2 in [1,2,3,4]
True
>>> a = tuple('abcd')
>>> a
('a','b','c','d')
>>> 'b' in a
True
>>> 'e' in a
False
```

Identity Operators

- To check if two values point to the same type of object, an identity operator `is` is used
- It returns a boolean value `True` if objects on either of its sides are the same otherwise returns `False`

```
>>> 1 is 1     # both are integer type
True
>>> 1 is 1.0   # both are not integer type
False
>>> 1 is 2     # both are integer type
True
>>> '1' is 1   # both are not integer type
False
```

Bitwise Operators

- Data are stored as bits in computers
- If we can operate directly on bits, we will have great flexibility and fast computation
- Bitwise operations find their use while dealing with hardware registers in embedded systems

Bitwise Operator	Description
>>	Bitwise left shift
<<	Bitwise right shift
&	Bitwise AND
|	Bitwise OR
~	Bitwise not

Bitwise Operators

- The binary composition of an `int` object can be shown using the `bin()`

```
>>> bin(1)
'ob1'
>>> bin(10)
'ob1010'
```

- When printing a binary representation of an integer number, 0b signifies that it is a binary representation

$$1010_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$
$$= 8 + 0 + 2 + 0 = 10_{10}$$

Bitwise Operators

- Bitwise operators operate on the number at the bit level
- So, the bitwise left shift operator will shift the value of bits one place to left
- Likewise, the bitwise right shift operator will shift the bit value one step to right
- This will result in a new binary representation and will be equivalent to a new decimal number

```
>>> bin(100)
'0b1100100'
>>> 100 << 1
200
>>> bin(200)
'0b11001000'
>>> 100 >> 1
50
>>> bin(50)
'0b110010'
>>> 100 << 2
400
>>> bin(400)
'0b110010000'
```

Precedence and Associativity

Operators	Associativity
() Highest precedence	Left - Right
**	Right - Left
+x , -x, ~x	Left - Right
*, /, //, %	Left - Right
+, -	Left - Right
<<, >>	Left - Right
&	Left - Right
^	Left - Right
|	Left - Right
Is, is not, in, not in, <, <=, >, >=, ==, !=	Left - Right
Not x	Left - Right
And	Left - Right
Or	Left - Right
If else	Left - Right
Lambda	Left - Right
=, +=, -=, *=, /= Lowest Precedence	Right - Left

input() Function

- To get some sort of input or information from the user, input() function is used

```
name = input("Enter your name : ")
print(name)
```

- Python takes all the input as a string input by default
- To convert it to any other data type we have to convert the input explicitly.

```
# Taking input from the user as integer
num = int(input("Enter a number: "))
add = num + 1

# Output
print(add)
```

input() Function

- we can take multiple inputs of the same data type at a time in python, using map() method in python.

```
a, b, c = map(int, input("Enter the Numbers : ").split())
print("The Numbers are : ",end = " ")
print(a, b, c)
```

Output Function

- Python provides the `print()` function to display output to the standard output devices.

```
1       a = 1.23
2       b = 4.56
3       print("a + b = ", a + b)
4       print("The summation of ", a , " and ", b , " = ", a + b)
5       c = a + b
6       print("The summation of %2.2f and %2.2f = %2.2f" %(a,b,c))
```

Output Function

```
1   # Initializing variables
2   a = 20
3   b = 10
4
5   # addition
6   sum = a + b
7
8   # subtraction
9   sub = a- b
10
11  # Output
12  print(f'The value of a is {a} and b is {b}')
13  print('The value of a is {} and b is {}'.format(a,b))
14  print('{2} is the sum of {0} and {1}'.format(a,b,sum))
15  print('{sub_value} is the subtraction of {value_a} and \
16        {value_b}'.format(value_a = a, value_b = b, sub_value = sub))
```

Output

```
The value of a is 20 and b is 10
The value of a is 20 and b is 10
30 is the sum of 20 and 10
10 is the subtraction of 20 and 10
```