Python Basics

Panchatcharam M

DATA TYPES

Numeric Types

```
>>> a = 5
>>> type(a)
<class ‘int’>
```

```
>>> a = 8.0
>>> type (a)
<class ‘float’>
```

```
>>> a = 5+4j
>>> type(a)
<class ‘complex’>
```

String Data Types

- Strings
- Can use “” or ‘’ to specify.

“abc” ‘abc’ (Same thing.)

- Unmatched can occur within the string
- “matt’s”
- Use triple double-quotes for multi-line strings or strings that contain both ‘and “ inside of them:
- “““a‘b“c”””

- >>> a = “Placement”
- >>> type (a)
- <class ‘str’>

Boolean Datatypes

```
>>> is_pass = True
>>> type (is_pass)
<class ‘bool’>
```

```
>>> is_pass = False
>>> type (is_pass)
<class ‘bool’>
```

List and Tuple Datatypes

```
>>> company= [“Google”,”Facebook”,”Apple”]
 >>> type (company)
<class ‘list’>
company=list(("Google","Facebook","Apple"))
```

```
>>> animals= (“Lion”,”Tiger”,”Cat”)
 >>> type (animals)
<class ‘tuple’>
animals= tuple((“Lion”,”Tiger”,”Cat”))
```

Set Datatypes

```
>>> company= {“Google”,”Facebook”,”Apple”}
 >>> type (company)
<class ‘set’>
company=set(("Google","Facebook","Apple"))
```

```
>>> animals= frozenset({“Lion”,”Tiger”,”Cat”})
 >>> type (animals)
<class ‘frozenset’>
animals= frozenset((“Lion”,”Tiger”,”Cat”))
```

Range and dict Datatypes

```
>>> x=range(10)
 >>> type (x)
<class ‘range’>
```

```
>>> s= {“name”:”Raj”,”age”:20,”Marks”:94.5,”Pass”:True}
 >>> type (s)
<class ‘dict’>
```

Byte Datatypes

```
>>> x=b”Placement”
 >>> type(x)
<class ‘bytes’>
```

```
>>> x=bytearray(5)
 >>> type(x)
<class ‘bytearray’>
```

```
>>> x=memoryview(x)
 >>> type(x)
<class ‘memoryview’>
```

Operators and Precedence

Arithmetic Operators

- **expression:** A data value or set of operations to compute a value.
- Examples: `1 + 4 * 3`

- **Arithmetic operators** we will use:

`+`	addition
`-`	subtraction/negation
`*`	multiplication
`/`	division
`%`	modulus, a.k.a. remainder
`**`	exponentiation
`//`	floor division

Precedence

- Order in which operations are computed.
- `**` has higher precedence than `*  /  //  %`
- `*  /  //  %` have a higher precedence than `+  -`

  `1 + 3 * 4` is `13`
- Parentheses can be used to force a certain order of evaluation.

  `(1 + 3) * 4` is `16`

- Multiple operators of same precedence
  - `**` right to left associativity
  - `*  /  //  %  +  -`left to right associativity

- **Advice:** Better use parentheses if you have more than one operators of multiple precedence

Expressions

- Python can also manipulate real numbers.

  Examples: `6.022` `-15.9997` `42.0` `2.143e17`

- The operators `+ - * / // % ** ( )` all work for real numbers.
- Example for `/` `15.0 / 2.0` is `7.5`
- Example for `//` `:15.0 / 2.0` is `7`
- The `%` produces an exact answer: `7.5 / 2.0` is `1.5`

- The same rules of precedence also apply to real numbers:

  Evaluate `( )` before `* / %` before `+ -`

- When integers and reals are mixed, the result is a real number.

  Example: `1 / 2.0` is `0.5`

Math Methods

Method	Description
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1***`, `***value2***`)`	larger of two values
`min(`***value1***`, `***value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

Constant	Description
`e`	2.7182818...
`pi`	3.1415926...
`tau (2*pi)`	6.2831853...

- To use many of these Methods, you must write the following at the top of your Python program:
- `import math`

Reserved Words

- Names are case sensitive and cannot start with a number.
- They can contain letters, numbers, and underscores.

```
bob     Bob     _bob     _2_bob_     bob_2     BoB
```

```
and,    assert,    break,    class,    continue,
def,    del,    elif, else,    except,    exec,
finally,    for,    from,    global,    if, import,
in,    is,    lambda,    not,    or,    pass,    print,
raise, return,    try,    while
```

Assignment: What goes on behind the scene

Assignment statement

- **Assignment Statement**: Stores a value into a variable.
  - Syntax:

    name = ***value***
  - Examples:

    ```
    x = 5
    gpa = 3.14
    x, y = 2,3
    ```
  - A variable that has been given a value can be used in expressions.

    `x + 4` is `9`

- **Exercise:** Evaluate the quadratic equation for a given *a*, *b*, and *c*.
  - $ax^2 + bx + c$

Assignment statement

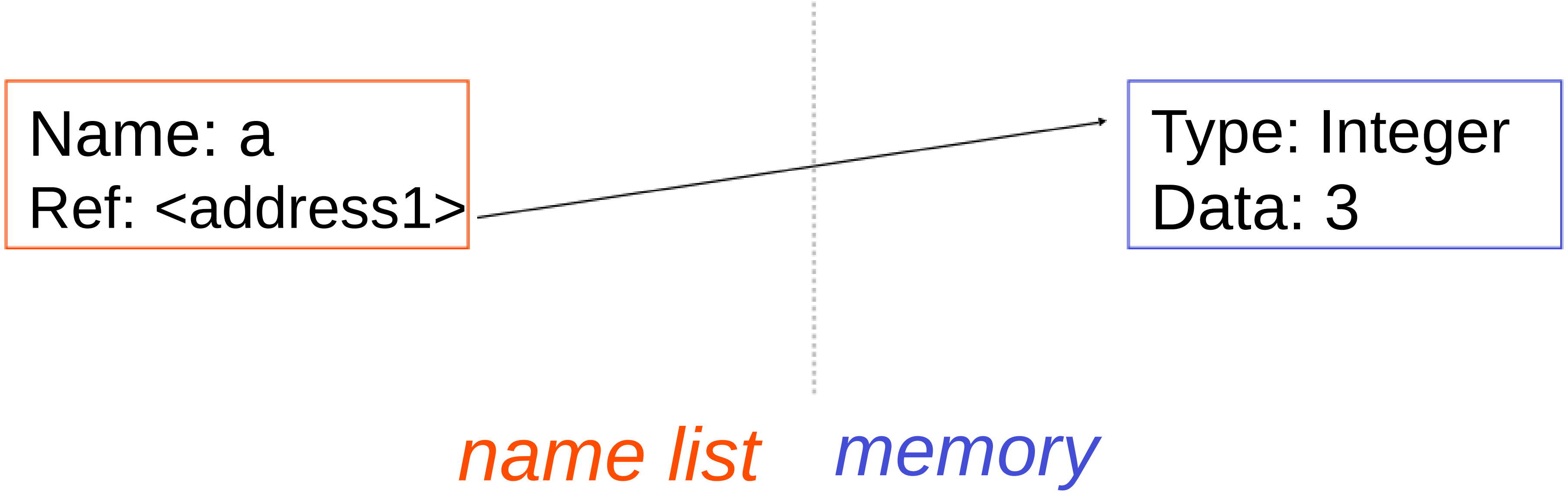
- **Assignment manipulates references**.
  - x = y **does not make a copy** of the object y references
  - x = y makes x **reference** the object y references

    name = ***value***
  - Examples:

```
a = [1,2,3] # a now references the list [1,2,3]
b = a        # b now references what a references
a.append(4)  # this changes the list a references
print(b)      # if we print b
[1,2,3,4]     # SURPRISED!?
```

Assignment statement

- **There is a lot going on when we type** `a = 3`
- **First, an integer *3* is created and stored in memory**
- **A name *a* is created**
- **A *reference* to the memory location storing the *3* is the assigned to the name *a***
- **So: When we say that the value of *a* is *3***
- **we mean that *a* now refers to the integer *3***



Name: a
Ref: <address1>

Type: Integer
Data: 3

name list *memory*

Assignment statement

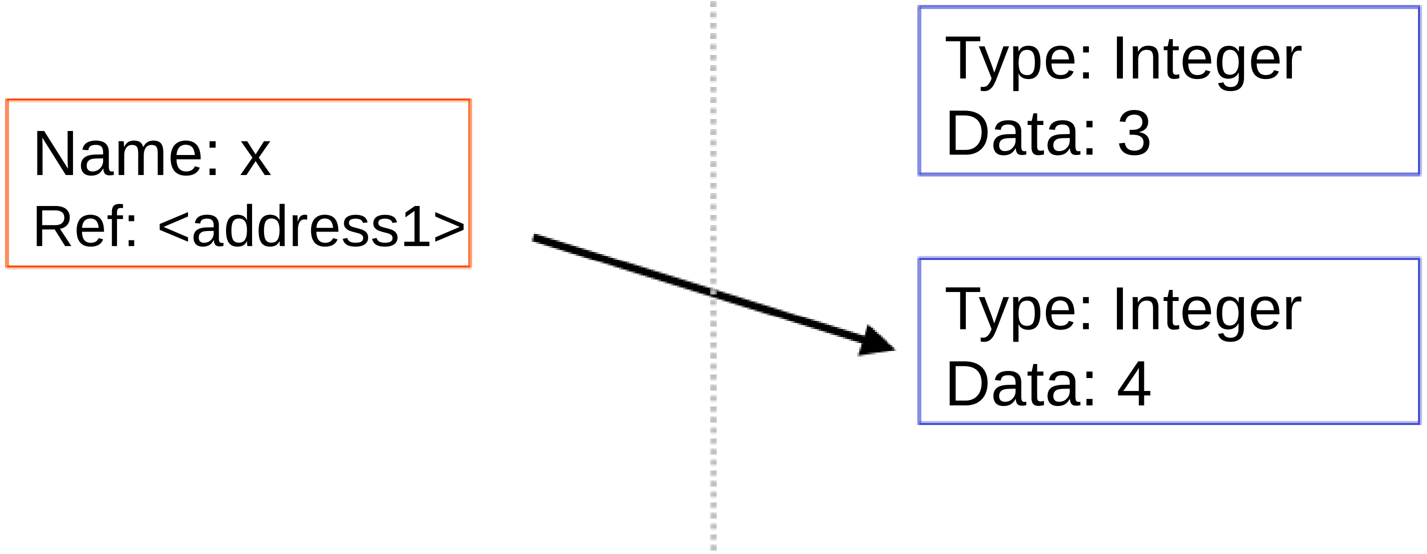
- The data 3 we created is of type integer. In Python, the datatypes integer, float, and string (and tuple) are “immutable.”
- This doesn’t mean we can’t change the value of x, i.e. *Change what x refers to* …
- For example, we could increment x

```
>>> x = 5
>>> x = x + 1
>>> print(x)
6
```

Assignment statement

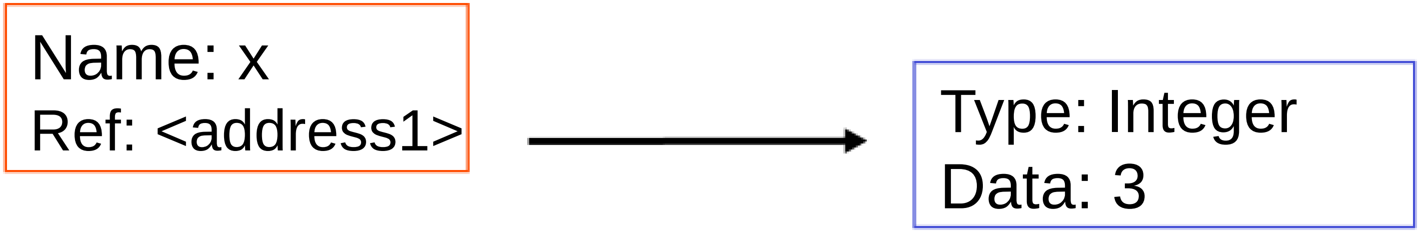
- **If we increment x, then what's really happening is:**
- The reference of name ***x*** is looked up.
- The value at that reference is retrieved
- The 3+1 calculation occurs, producing a new data element **4** which is assigned to a fresh memory location with a new reference.
- *The name **x** is changed to point to this new reference.*
- *The old data **3** is garbage collected if no name still refers to it*

```
>>> x = x + 1
```



Assignment statement

```
→ >>> x = 3          # Creates 3, name x refers to 3
  >>> y = x          # Creates name y, refers to 3.
  >>> y = 4          # Creates ref for 4. Changes y.
  >>> print(x)       # No effect on x, still ref 3.
  3
```

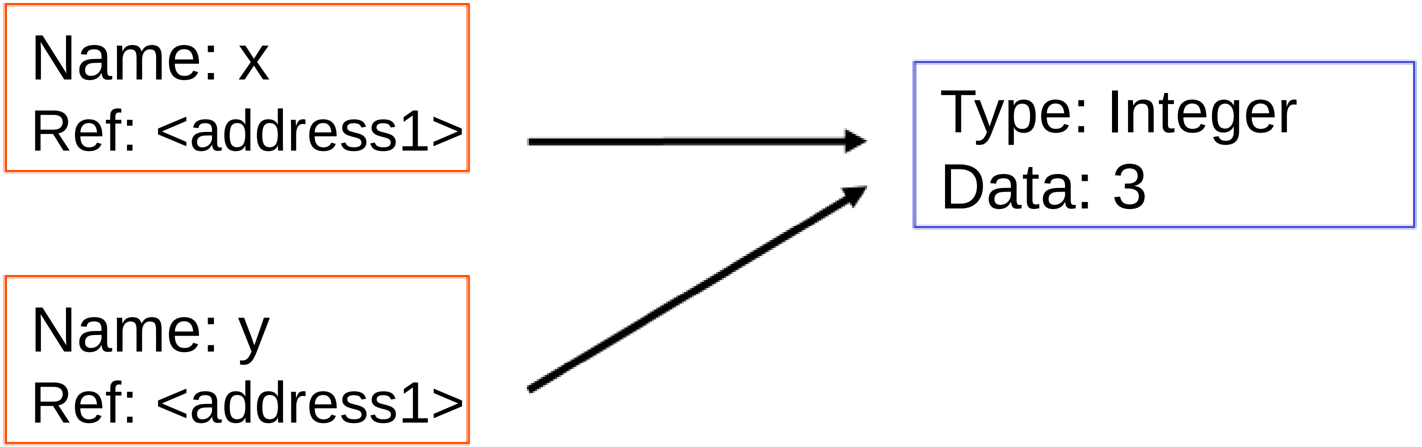


Name: x
Ref: <address1>

Type: Integer
Data: 3

Assignment statement

```
→ >>>  x  =  3          #  Creates  3,  name  x  refers  to  3
→ >>>  y  =  x          #  Creates  name  y,  refers  to  3.
  >>>  y  =  4          #  Creates  ref  for  4.  Changes  y.
  >>>  print(x)         #  No  effect  on  x,  still  ref  3.
  3
```



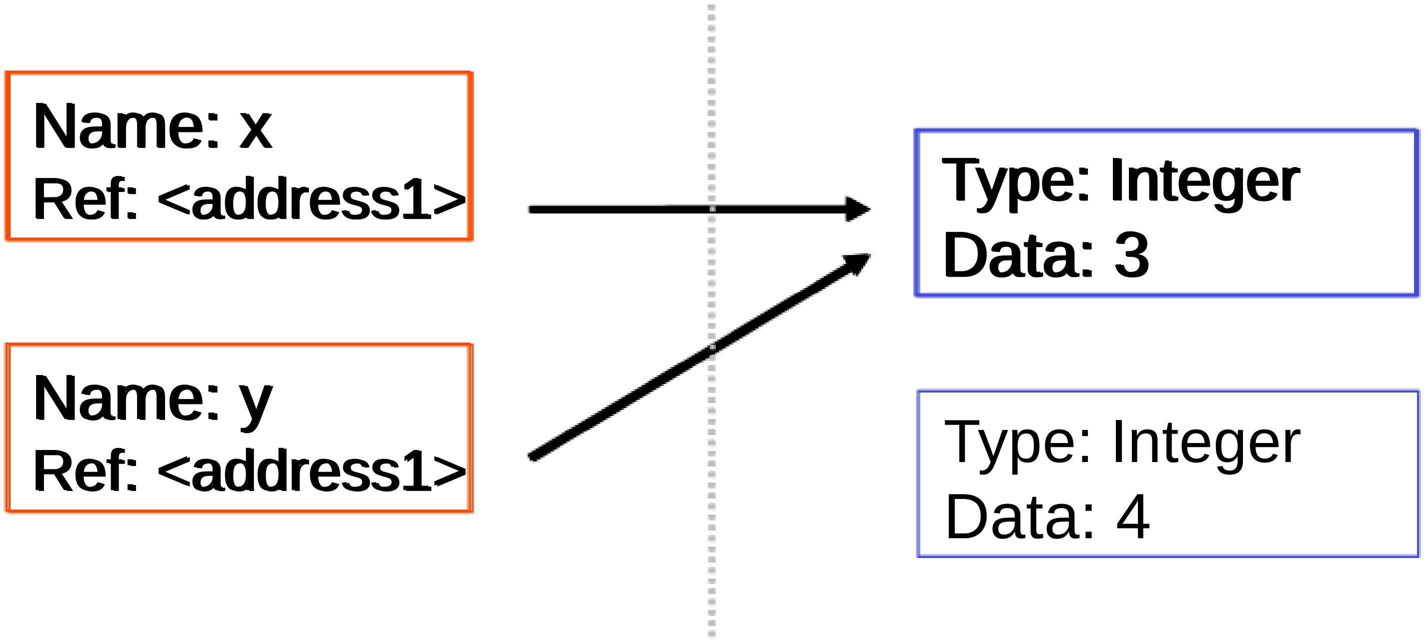
Name: x
Ref: <address1>

Name: y
Ref: <address1>

Type: Integer
Data: 3

Assignment statement

```
→ >>>  x  =  3          #  Creates  3,  name  x  refers  to  3
→ >>>  y  =  x          #  Creates  name  y,  refers  to  3.
→ >>>  y  =  4          #  Creates  ref  for  4.  Changes  y.
  >>>  print(x)         #  No  effect  on  x,  still  ref  3.
  3
```



Name: x
Ref: <address1>

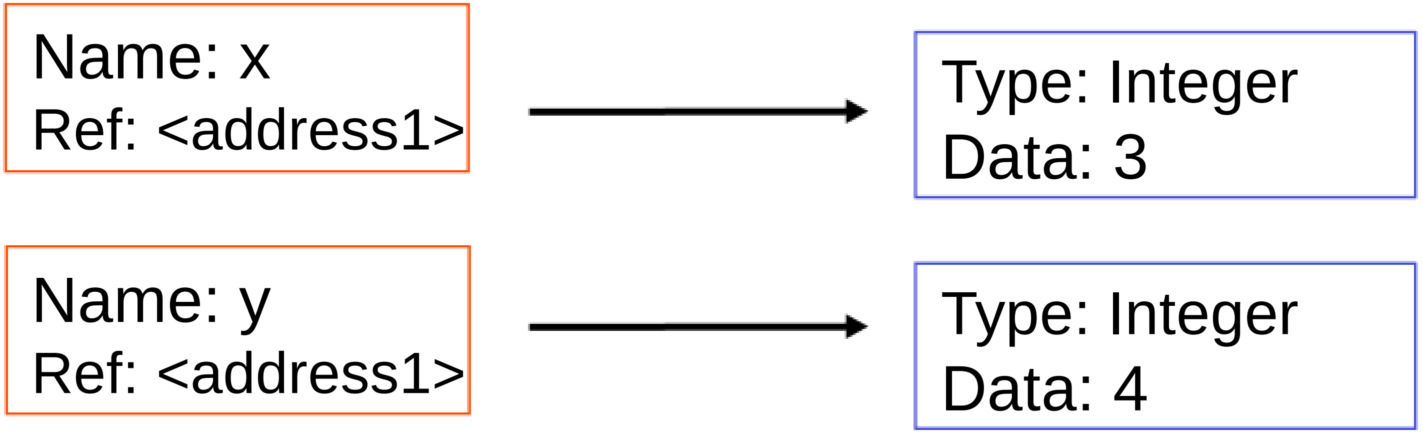
Name: y
Ref: <address1>

Type: Integer
Data: 3

Type: Integer
Data: 4

Assignment statement

```
→ >>>  x = 3          # Creates 3, name x refers to 3
→ >>>  y = x          # Creates name y, refers to 3.
→ >>>  y = 4          # Creates ref for 4. Changes y.
  >>>  print(x)       # No effect on x, still ref 3.
  3
```



Name: x
Ref: <address1>

Type: Integer
Data: 3

Name: y
Ref: <address1>

Type: Integer
Data: 4

Assignment statement

- ❑ **For other data types (lists, dictionaries, user-defined types), assignment works differently.**
- ❑ These datatypes are **“mutable.”**
- ❑ When we change these data, we do it *in place.*
- ❑ We don’t copy them into a new memory address each time.
- ❑ If we type y=x and then modify y, both x and y are changed

immutable

```
>>> x = 3
>>> y = x
>>> y = 4
>>> print(x)
3
```

mutable

x = some mutable object

y = x

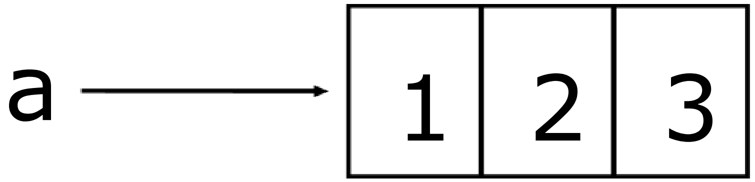
make a change to y

look at x

x will be changed as well

Sharing List

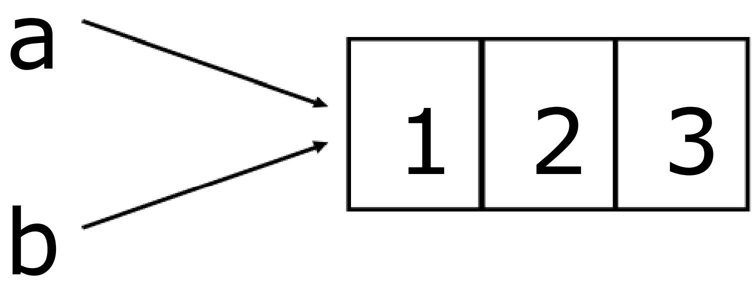
```
a = [1, 2, 3]
```



a

1 2 3

```
b = a
```

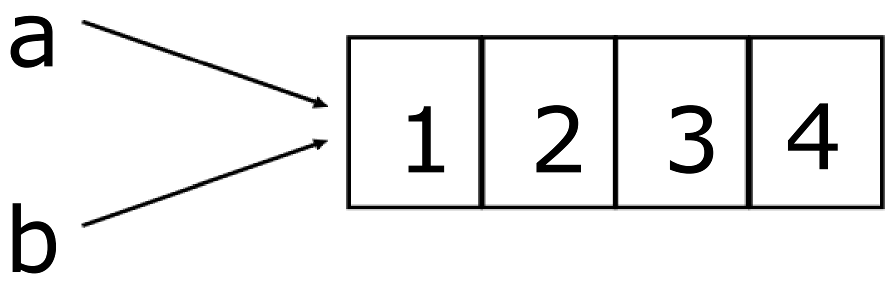


a

b

1 2 3

```
a.append(4)
```



a

b

1 2 3 4

PRINT AND INPUT

print

- `print` : Produces text output on the console.
- Syntax:

  `print("`***Message***`“)`

  `print(`***Expression)***

  - Prints the given text message or expression value on the console, and moves the cursor down to the next line.

  `print(`***Item1 , Item2 , … , ItemN)***

  - Prints several messages and/or expressions on the same line.

print

- **Examples:**

```
print("Hello, world!")
age = 45
print("You have", 65 - age, "years until retirement")
```

Output:

```
Hello, world!
You have 20 years until retirement
```

input

- `input` : Reads a number from user input.
  - You can assign (store) the result of `input` into a variable.
  - **Example:**

```
age = input("How old are you? ")
print("Your age is", age)
print("You have", 65 - int(age), "years
until retirement")
```

Output:

```
How old are you? 53
Your age is 53
You have 12 years until retirement
```

MORE ON PRINT

FANCY OUTPUT

print

- `print` : Produces text output on the console.

- Full Syntax:

```
print(*objects, sep=' ', end='\n',
file=sys.stdout, flush=False)
```

  - **objects:** objects to be printed
  - **sep:** object separated by sep
  - **file:** with write string method
  - **flush:** stream is forcibly flushed

print

- **Examples:**

```
a,b = 10,5
print("a = ", a, sep='00000', end='\n\n\n')
print(“b = ", a, sep='0', end='')
```

Output:

```
a = 0000010



a = 05
```

print

- **Examples:**

```
fp=open('Testing.txt','w')
print("MA522M-Data Science Programming Laboratory",file=fp)
fp.close()
```

Formatted string in print

- **Examples:**

```
import math
print(f'The value of pi is approx {math.pi:.3f}.')
print('There are {}, {}, {}, {} in chess'.format('knights',
'king','queen','horses'))
FirstName='Raja'
LastName='Kumar'
Marks=43.5
print('Student Name {1} {2}. His Mark is {0}'.format(Marks,FirstName, LastName))
print('Student Name {0} {1}. His Mark is {2}'.format(FirstName, LastName,Marks))
print('Student Name {first} {last}. His Mark is
{mark}'.format(mark=Marks,first=FirstName, last=LastName))
print('Student Name {first} {last}. His Mark is {mark}'.format(first=FirstName,
mark=Marks,last=LastName))
```

Formatted string in print

- **Examples:**

```
x=2
print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
x=3
print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
x=8
print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
x=10
print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
```

More on Lists

List: Key Functions

```
x=[1,2,3,4,5]
print('First Element: ',x[0]) #indexing
print(‘Second Element: ',x[1])
print('Last Element: ',x[-1]) #indexing from last
print('Slicing: ',x[2:]) #slicing
print('Slicing 2: ',x[-2:]) #slicing
x=x+[2,3,4,5,6] #concatenation
print('Concatenation:',x)
x.append(7) #appending
print('Appended 7: ',x)
x.remove(2) #removing
print('Removed 2: ',x)
x[2]=7 #replacing
print('Replacment: ',x)
```

List: Key Functions

```
y=[0,0,0] #nested list
y[0]=[1,2,3] #replacing
y[1]=[3,4,5]
y[2]=[5,6,7]
print('Nested List :',y)
#can be mixture of all data types
A=['a','b','c','d','e','f',0,0.1,"Ram",True,3+4j,[1,2,3,4]]
print('Mixtures: ',A)
print('Slicing Again: ',A[2:7]) #slicing
#Remove item
A[2:6]=[]
print('Removing again: ',A)
del A[2]
print('Deleting: ',A)
#Remove all
A[:]=[]
print('Removed Everything: ',A)
```

List: Key Functions

```
#Length
A=['a','b','c','d','e','f’]
print(‘A*2:’,A*2)
print('New List:',A)
print('Length of List: ',len(A))
A.clear()
print('Cleared Again: ',A)
A=['a','b','c','d','e','f']
A.extend('g')
print('Extended',A)
```

```
x=x+[2,3,4,5,6]
print(x)
print('Number of occurences of 3: ',x.count(3))
#number of occurences
print('Index of 5 in the list: ',x.index(5))
print('Maximum and Minimum: ',max(x),min(x))
#minimum and maximum
x.reverse()
print('Reversed: ',x)
x.sort()
print('Sorted: ',x)
reversed(x)
print(x)
```

MORE ON TUPLES

Tuples: Key Functions

```
heros=('Arthos','Porthos','Aramis','Romeo','Juliet')
print(heros)
print(len(heros))
print(heros[1],heros[2],heros[-1],heros[-
2],heros[2:],heros[-2:],sep=' & ')
print(heros.index('Porthos'))
```

MORE ON SETS

Sets: Key Functions

```
emptyset=set()
print(emptyset)
print(x)
numbers=set(x)
print(numbers)
y=list(numbers)
print(y)
```

```
programming=set(['C','C++','Python','Ruby','Java','Scala','Swift','Perl'])
print(programming)
programming.add('Python')
print(programming)
programming.add('SQL')
print(programming)
compilers=set(['C','C++','Scala'])
interpreters=set(['Python','Java'])
programming.update(compilers)
```

Sets: Key Functions

```
intersect=compilers.intersection(interpreters)
print(intersect)
union=compilers.union(interpreters)
print(union)
union=union.union(programming)
print(union)
diff=programming.difference(compilers)
print(diff)
print(compilers.isdisjoint(interpreters))
print(compilers.issubset(programming))
print(programming.issuperset(compilers))
print(programming^compilers)#symmetric difference
print(programming.symmetric_difference(compilers))
```

More on Dictionaries

dict: Key Functions

```
course={‘MA612L':'PDE',’MA522M’:’Data Science Programming Laboratory',’MA502L':'DE',’MA633L':'Numerical'}
print(course)
print(course[‘MA612L'])
print(course.get(‘MA522M'))
print(course.keys())
print(course.values())
print(course.items())
```

```
course['MA101’]=‘EM-1'
print(course)
print(len(course))
course.clear()
print('cleared: ',course)
```

dict: Key Functions

```
btechcourse={'MA2021':'Linear Algebra','MA2022':'Complex Methods','MA2023':'Probability'}
print(btechcourse)
course.update(btechcourse)
print(course)
herodictionary=dict.fromkeys(heros)
print(herodictionary)
herodictionary=dict.fromkeys(heros,[1,2,3,4,5])
print(herodictionary)
```