Building Blocks

- Values and Variables
  - numeric values
  - variables
  - assignment
  - identifiers
  - reserved words
  - Comments

Standard Data Types

- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types
  - Numbers
  - String
  - List
  - Tuple
  - Dictionary

Python Numbers

- Number data types store numeric values.
- Number objects are created when you assign a value to them. For example –

```
var1 = 1
var2 = 10
```

- You can also delete the reference to a number object by using the del statement. The syntax of the del statement is –

```
del var1[,var2[,var3[....,varN]]]
```

- You can delete a single object or multiple objects by using the del statement. For example –

```
del var
del var_a, var_b
```

Assigning Values to Variables

- Python variables do not need explicit declaration to reserve memory space.
- The declaration happens automatically when you assign a value to a variable.
- The equal sign (=) is used to assign values to variables.
- For example –

```
counter = 100          # An integer assignment
miles   = 1000.0       # A floating point
name    = “Nielit"       # A string
print(counter)
Print(miles)
Print(name)
```

- Here, 100, 1000.0 and “Nielit" are the values assigned to *counter*, *miles*, and *name* variables, respectively.

Expressions

- 3+4

7

- 3+4+5

12

- Print(4+5+6)

15

Quotes

- If we write
- “19” or ‘19’, it is an example of a string value and not numeric.
- A string is a sequence of characters. Strings most often contain non-numeric characters.
- Python recognizes both single quotes (’) and double quotes (") as valid ways to delimit a string value. If a single quote marks the beginning of a string value, a single quote must delimit the end of the string and same for double quotes, one cannot mix.
- If you wish to embed a single quote mark within a single-quote string, you can use the backslash to escape the single quote (\’). An unprotected single quote mark would terminate the string. Similarly, you may protect a double quote mark in a double-quote string with a backslash (\").

```
print("Did you know that 'word' is a word?")
print(‘Did you know that \’word\’ is a word?’)
```

Type function

- It is important to note that the expressions 4 and ’4’ are different. One is an integer expression and the other is a string expression.
- All expressions in Python have a type. The type of an expression indicates the kind of expression it is. An expression’s type is sometimes denoted as its class. At this point we have considered only integers and strings.
- The built in type function reveals the type of any Python expression
- >>type(4)
- <class ‘int’>
- >>type(‘4’)
- <class ‘str’>

Python numbers

- The built in int function converts the string representation of an integer to an actual integer, and the str function converts an integer expression to a string:
- Str(4)
- Int(‘5’)
- The expression str(4) evaluates to the string value ’4’, and int(’5’) evaluates to the integer value 5.
- Any integer has a string representation, but not all strings have an integer equivalent:

Strings and numerics

```
>>str(1024)
’1024’

>>int(’hello’)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: ’hello’

>>int(’3.4’)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: ’3.4’
```

Integer Values

- 4 is an integer value. Integers are whole numbers, which means they have no fractional parts, and they can be positive, negative, or zero. Examples of integers include 4, −19, 0, and −1005. In contrast, 4.5 is not an integer, since it is not a whole number.

- The Python statement

print(4)

- prints the value 4.

The plus operator (+)

>>5 + 10

15

>>’5’ + ’10’

’510’

>>’abc’ + ’xyz’

’abcxyz’

>> ’5’ + 10

Reproduces error

type function

```
>>type(4)
class ’int’>
>>type(’4’)
<class ’str’>
>>type(4 + 7)
<class ’int’>
>>type(’4’ + ’7’)
<class ’str’>
>>type(int(’3’) + int(4))
<class ’int’>
```

Variables and Assignment

X=10

Print(x)

Prints value of x and Print(‘x’) prints the message x

5 = x

Reproduces error

File "<stdin>", line 1

SyntaxError: can’t assign to literal

To treat x as a string, the + operator will use string concatenation:

Print(“x=“ + str(x))

Prints x=10

Python Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object.
- An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
- Python does not allow punctuation characters such as @, $, and % within identifiers.
- Python is a case sensitive programming language. Thus, **Manpower** and **manpower** are two different identifiers in Python.
- **Reserved Words :** Python maintains a list of Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names.

Tuple assignment

- A tuple is a comma separated list of expressions.
  - **x, y, z = 100, -45, 0**
  - Print(‘x=‘, x, ’y=‘ ,y, ’z=‘,z)

- x, y, z is one tuple, and 100, -45, 0 is another tuple. Tuple assignment works as follows: The first variable in the tuple on left side of the assignment operator is assigned the value of the first expression in the tuple on the left side (effectively x = 100).

- Similarly, the second variable in the tuple on left side of the assignment operator is assigned the value of the second expression in the tuple on the left side (in effect y= -45). z gets the value 0.

User Input

- Print(“enter some text”)
- x=Input()
- Print(“text entered:”, x)
- Print(‘Type of x’, type(x))
- The input function produces only strings, but we can use the int function to convert a properly formed string of digits into an integer.

  Example

  - Print(“enter first number”)
  - X=input()
  - Print(“enter second number”)
  - Y=input()
  - num1=int(x)
  - Num2=int(y)
  - print(num1, '+', num2, '=', num1 + num2)

Better Way

- Since user input almost always requires a message to the user about the expected input, the input function optionally accepts a string that it prints just before the program stops to wait for the user to respond. The statement
- **x = input('Please enter some text: ')**

Example

- X=input('**Please enter some text: **')
- Y=input('**Please enter some text: **')
- num1=int(x)
- Num2=int(y)
- **print(num1, '+', num2, '=', num1 + num2)**

Still Better Way

Example

- num1=int(input('**Please enter some text: **'))
- num2=int(input('**Please enter some text: **'))
- **print(num1, '+', num2, '=', num1 + num2)**

The eval Function

- The input function produces a string from the user's keyboard input. If we wish to treat that input as a number, we can use the int or float function to make the necessary conversion:
- **x = float(input('Please enter a number'))**
- **But using eval**
  - **x1=eval(input('Entry x1? '))**
  - **print('x1 =', x1, ' type:', type(x1))**
- When the user enters 4, the variable's type is integer. When the user enters 4.0, the variable is a floating-point variable. For x3, the user supplies the string 'x3' (note the quotes), and the variable's type is string. The more interesting situation is x4. The user enters x1 (no quotes). The eval function evaluates the non-quoted text as a reference to the name x1.

Eval continued

- The eval function dynamically translates the text provided by the user into an executable form that the program can process.
- This allows users to provide input in a variety of flexible ways; for example, users can enter multiple entries separated by commas, and the eval function evaluates it as a Python tuple.
  - **num1, num2 = eval(input('Please enter number 1, number 2: ‘**
  - **print(num1, '+', num2, '=', num1 + num2)**

Note: The user now must enter the two numbers at the same time separated by a comma:

- **print(eval(input()))**
- The users enters the text 4 + 10, and the program prints 14.

Controlling the print function

- The print statement accepts an additional argument that allows the cursor to remain on the same line as the printed text:

- **print('Please enter an integer value:', end='')**

The statement **print('Please enter an integer value:')** is an abbreviated form of the statement

- **print('Please enter an integer value:', end='\n')**
- Similarly, the statement
-
- **print()**
-
- is a shorter way to express
-
- **print(end='\n')**

Controlling the print function

- **w, x, y, z = 10, 15, 20, 25**
- Print(w,x,y,z)
  10 15 20 25
- Print(w,xy,y,z,sep=‘,’)
  10,15,20,25
- Print(w,xy,y,z,sep=‘:’)
  10:15:20:25
- Print(w,xy,y,z,sep=‘’)
  10152025
- Print(w,xy,y,z,sep=‘----’)
  10----15----20----25

Expressions and Arithmetic

x + y	x added to y, if x and y are numbers
	x concatenated to y, if x and y are strings
x - y	x take away y, if x and y are numbers
x * y	x times y, if x and y are numbers
	x concatenated with itself y times, if x is a string and y is an integer
	y concatenated with itself x times, if y is a string and x is an integer
x / y	x divided by y, if x and y are numbers
x // y	Floor of x divided by y, if x and y are numbers
x % y	Remainder of x divided by y, if x and y are numbers
x ** y	x raised to y power, if x and y are numbers

Modulus Operator

When we apply the +, -, *, //, %, or ** operators to two integers, the result is an integer. The / operator applied to two integers produces a floating-point result.

- **print(10/3, 3/10, 10//3, 3//10)**

  Prints 3.3333333333333335 0.3 3 0

Floating-point arithmetic always produces a floating-point result.

- **print(10.0/3.0, 3.0/10.0, 10.0//3.0, 3//10.0)**

  Prints 3.3333333333333335 0.3 3.0 0.0

The modulus operator (%) computes the remainder of integer division; thus,

- **print(10%3, 3%10)**

  Prints 1 3

Operator Precedence and Associativity

- The operators in each row have a higher precedence than the operators below it. Operators within a row have the same precedence.

Unary	+, -	
Binary	*, /, %	Left
Binary	+, -	Left
Binary	=	Right

Reserved Words

	and	del	from	None	try	
	as	elif	global	nonlocal	True	
	assert	else	if	not	while	
	break	except	import	or	with	
	class	False	in	pass	yield	
	continue	finally	is	raise		
	def	for	lambda	return		

Reserved Words

- We are free to reassign the reserved names and use them as variables. It generally is not a good idea to do so.
- Example :
- print(’Our good friend’)

  Our good friend
- print

  <built-in function print>
- type(print)

  <class ’builtin_function_or_method’>
- print = 77
- print

  77
- print(’Our good friend’)

  Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  TypeError: ’int’ object is not callable

- type(print)

  <class ’int’>

Assigning function names

- Not only can a function name can be reassigned, but a variable can be assigned to a function.
-
- **my_print = print**
-
- **my_print('hello from my_print!') hello from my_print!**
-
- After binding my_print to print we can use my_print is exactly the same way as the built-in print function.

Floating Point Numbers

- The range of floating-points values (smallest **Floating Point Numbers**
- value to largest value, both positive and negative) and precision (the number of digits available) depends of the Python implementation for a particular machine.
- Unlike Python integers which can be arbitrarily large (or, for negatives, arbitrarily small), floating-point numbers have definite bounds.
- Floating-point numbers can be expressed in scientific notation.
- The number 6.022×10^{23} is written 6.022e23. The number to the left of the e (capital E can be used as well) is the mantissa, and the number to the right of the e is the exponent of 10.
- As another example, -5.1×10^{-4} is expressed in Python as -5.1e-4. Listing 2.7 (scientificnotation.py) prints some scientific constants using scientific notation.

More Arithmetic Operators

- **x += 1**
- **x -= 1**
- x is a variable.
- op= is an arithmetic operator combined with the assignment operator; for our purposes, the ones most useful to us are +=, -=, *=, /=, //=, and %=.
- exp is an expression compatible with the variable x.
- This means the statement
- **x *= y + z;**
-
- is equivalent to
- **x = x * (y + z);**
-
-

Lines and Indentation

- Python provides **no braces** to indicate blocks of code for class and function definitions or flow control.
- Blocks of code are denoted **by line indentation**, which is rigidly enforced.
- The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount.
- For example –

```
if True:
    print "True"
else:
    print "False“
```

Indentation Error

```
print(’hi’)
File "<stdin>", line 1
print(’hi’)
^
IndentationError: unexpected indent
```

Multi-line Statements

- Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example −

```
total = item_one + \
        item_two + \
        item_three
```

- Statements contained within the [], {}, or () brackets **do not need** to use the line continuation character.
- For example −

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

- The **semicolon (;)** allows multiple statements on the single line given that neither statement starts a new code block.
- Here is a sample snip using the semicolon −

```
import sys; x = 'NIELIT'; sys.stdout.write(x + '\n')
```

Comments in Python

- **Comments in Python**
- A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

```
# First comment
print "Hello, Python!"
```

- This produces the following result –

```
Hello, Python!
```

- You can type a comment on the same line after a statement or expression –

```
name = “NIELIT" # This is again comment
```

- You can comment multiple lines as follows –

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

Multiple Assignment

- Python allows you to assign a single value to several variables simultaneously. For example −

  a = b = c = 1

- You can also assign multiple objects to multiple variables. For example −

  a,b,c = 1,2,“Nielit"

- Here, two integer objects with values 1 and 2 are assigned to variables a and b respectively, and one string object with the value “Nielit" is assigned to the variable c.

Python Strings

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator. For example −

```
str = 'Hello World!'
print str          # Prints complete string
print str[0]       # Prints first character of the string
print str[2:5]     # Prints characters starting from 3rd to 5th
print str[2:]      # Prints string starting from 3rd character
print str * 2      # Prints string two times
print str + "TEST" # Prints concatenated string
```

- This will produce the following results − Hello World!, H, llo, llo World!, Hello World!Hello World!, Hello World!TEST

Conditional Execution

- The simplest Boolean expressions in Python are True and False. In a Python interactive shell we see:
- >>>True

True

>>>False

False

>>>type(True)

<class ‘bool’>

>>>type(False)

<class ‘bool’>

Python Relational Operators

Expression	Meaning
x == y	True if x = y (mathematical equality, not assignment); otherwise, false
x < y	True if x < y; otherwise, false
x <= y	True if x ≤ y; otherwise, false
x > y	True if x > y; otherwise, false
x <= y	True if x ≥ y; otherwise, false
x != y	True if x 6= y; otherwise, false

Some Relational Expressions

Expression	Value
10 < 20	True
10 >= 20	False
x < 100	True if x is less than 100
x != y	True unless x and y are equal

The simple if

The general form of the if statement is:

```
if  condition :
block
```

- The reserved word if begins a if statement.
- The condition is a Boolean expression that determines whether or not the body will be executed.
- A colon (:) must follow the condition.
- The block is a block of one or more statements to be executed if the condition is true. Recall that the statements within a block must all be indented the same number of spaces from the left.
- The block within an if must be indented more spaces than the line that begins the if statement. The block technically is part of the if statement. This part of the if statement is sometimes called the body of the if.

The simple if

- Python requires the block to be indented. If the block contains just one statement, some programmers will place it on the same line as the if; for example, the following if statement that optionally assigns y
- **if x < 10:**

  y = x

could be written

If x < 10: y = x

but may not be written as

if x < 10:

y = x

Inden-tation is how Python determines which statements make up a block

- Remember when checking for equality, as in
-
- **if x == 10:**
-
- **print('ten')**

The if/else Statement

```
Dividend,divisor=Eval(input(“Please enter two numbers to divide”))
If divisor != 0:
    Print(dividend, ‘/’, divisor, ‘=‘, dividend/divisor)
else:
    print(‘division by 0 is not allowed’)
```

Compound Boolean Expressions

- Python allows an expression like
-
- **x <= y and y <= z**
-
- which means x ≤ y ≤ z to be expressed more naturally:
-
- **x <= y <= z**
- Similarly, Python allows a programmer to test the equivalence of three variables as
-
- **if x == y == z:**
-
- **print('They are all the same')**

Assigning Bool Values

```
x = 10
y = 20
b = (x == 10)                    # assigns True to b
b = (x != 10)                    # assigns False to b
b = (x == 10 and y == 20)        # assigns True to b
b = (x != 10 and y == 20)        # assigns False to b
b = (x == 10 and y != 20)        # assigns False to b
b = (x != 10 and y != 20)        # assigns False to b
b = (x == 10 or y == 20)         # assigns True to b
b = (x != 10 or y == 20)         # assigns True to b
b = (x == 10 or y != 20)         # assigns True to b
b = (x != 10 or y != 20)         # assigns False to b
```

- **if x == 1 or 2 or 3:**

  print("OK")
- The code would always print the word OK regardless of the value of x. Since the == operator has lower precedence than ||, the expression
- The expression x == 1 is either true or false, but integer 2 and 3 are always interpreted as true.

1 <= x <= 3 also would work.

The correct statement would be

if x == 1 or x == 2 or x == 3:

  print("OK")

Multi-way Decision Statements

```
value = eval(input("Please enter an integer in the range 0...3: "))
If value < 0:
        print(“too small”)
Else:
        if value==0:
                print (“Zero”)
        else:
                if value==1:
                        print(“One”)
                else:
                        if value==2:
                                print(“Two”)
                        else:
                                if value==3:
                                        print(“Three”)
Print(“Finished”)
```

Note: See the indentation

Multi-way Decision Statements

- Python provides a multi-way conditional construct called if/elif/else that permits a more manageable textual structure for programs that must check many conditions.

```
If value < 0:
    print(“too small”)
elif value==0:
    print (“Zero”)
elif value==1:
    print(“One”)
elif value==2:
    print(“Two”)
elif value==3:
    print(“Three”)
```

Multi-way Decision Statements

- The word elif is a contraction of else and if; if you read elif as else if, you can see how the code fragment

```
else:
if value == 2:
    print("two")
```

- Can be rewritten as

```
elif value == 2:
print("two")
```

Conditional Expression

- As purely a syntactical convenience, Python provides an alternative to the if/else construct called a conditional expression.

- The general form of the conditional expression is

$expression_1$ if condition else $expression_2$

Example :

c = d if a != b else e

Would mean

```
If a !=b
    c=d
Else
    c=e
```

Conditional Expression

- Example

```
Dividend,divisor=Eval(input(“Please enter two numbers to divide”))
If divisor != 0:
        Print(dividend, ‘/’, divisor, ‘=‘, dividend/divisor)
else:
        print(‘division by 0 is not allowed’)
```

- Can be written as

```
Dividend,divisor=Eval(input(“Please enter two numbers to divide”))
show=dividend/divisor if divisor !=0 else ‘Error, cannot divide by zero’Print(show)
```

Exercise

- Write a Python program that requests five integer values from the user. It then prints the maximum and minimum values entered. If the user enters the values 3, 2, 5, 0, and 1, the program would indicate that 5 is the maximum and 0 is the minimum. Your program should handle ties properly; for example, if the user enters 2, 4 2, 3 and 3, the program should report 2 as the minimum and 4 as maximum.
- Write a Python program that inputs a two digit number , example 35 and prints :
  - Three Five
  - Hint: Use String slicing