Python

- Invented in 1989-90 by Guido van Rossum. It has strong support in terms of libraries (called modules). Python 3 was created in 2008 to remove problems with Python 2. However, it is incompatible with Python 2 and versions 2 and 3 are still being used simultaneously. Now most earlier libraries have been ported to version 3 and almost all new resources are being developed in Python 3. We will use version 3 in this course. The latest version is 3.7.0. But we don't need the latest version for this course 3.5.x will do. Bug fixed releases of earlier versions are released. However, version 2 will remain at 2.7.x
- Python code (like Java) compiles to bytecodes and is interpreted. Many IDEs (Integrated Development Environments) are available. It can also interface with C, C++ and Java. It can also connect to several widely used databases.
- Downloads, tutorials, documentation and other Python resources are available at `www.python.org`. It is open source.

Values in Python

- **Objects** are the basic entities or values in Python.
- Each object has a type. Types can be *scalar* or *non scalar*. Values with scalar type are atomic or indivisible. They are not 'literally' indivisible but one should avoid breaking them up. Scalar objects are:
  - Integers. They have int type
  - Floating point numbers. They have float type
  - Boolean values `True` and `False`. They have bool type.
  - The unique value `None`. This value is used to signal empty or null. For example, if a function exits without return being executed then `None` is returned. Do not confuse it with undefined. If a variable has not been bound it is undefined. To be bound to `None` it should be explictly assigned `None`. The type of `None` is NoneType.
- Non scalar values have internal structure - they are composed of other scalar or non scalar values. For example, lists, tuples, strings, instances of classes etc. We can extract parts or components of such values. For example, we can ask for the first or last element of a list.

Operations on scalar values of type int, float

- The standard binary operators +, -, * work on values of type int or float.If any value is a float then the result is a float else it is int.
- / is the float division operator and always produces a float value. So, 5/2 is 2.5.
- // is the integer division operator and produces the integer quotient. If either operand is a float then the quotient is returned as a float value. So, 5//4 is 1 and 5.0//4 is 1.0.
- % is the remainder operator. Both operands are supposed to be int values. But if either value is float the result will be a float but cannot be relied on. Use only integer operands with %.
- ** is the exponentiation operator. If both operands are int values the result is an int else a float.
- The binary comparison operators are: ==(equal), ! =(not equal), <(less), >(greater), <=(less or equal), >=(greater or equal).
- Precedence from high to low is: **, (*,/,//), (+,-). Brackets '(',')' override precedence.

Operators on scalar values of type bool

- If x, y are bound to values of type bool then (x `and` y) is `True` only if both x and y are `True` else it is false. (x `or` y) is `False` only if both x and y are `False` else it is true. (`not` x) is `True` if x is `False` and is `False` if x is `True`.

Variables and assignment operator

- Variables act as handles or names for values. They do not have a specific type. Only the values associated with a variable have a type. The same variable can be associated with objects of different types at different times. This is unlike other popular languages like C, C++, Java. At any point in time a variable is bound to at most one value. On the other hand the same object can be bound to different variables.
- Variables can be bound to an object by the assignment operator $=$. If a variable has not been assigned then its value is not defined and using it in any expression will lead to error.
- Variable names must start with a letter and can contain letters, digits and _. Names are case sensitive so x and X are different variables. There is a fixed set of keywords like *if*, *else*, *for*, *import*, *in* etc. that are reserved and cannot be a variable name.

Variables and multiple assignment

- Python allows multiple assignment. For example,

  `x,y,z=3,4,5`

  will bind 3 to x, 4 to y, 5 to z.

Comments

- Python has only single line comments. The text after # till end of line is treated as a comment.
- For multi-line comments one must put a # at the beginning of each line. Most editors/IDEs have a simple way to do this. Do not use a pair of triple quotes """ - that is meant only for documentation text.

Conditionals and indenting

- Conditional code execution for if-then, if-then-else, if-then-elif-else kind of conditions. This is best illustrated by examples. Consider Python programs for a) Print x if x is greater than 9 b) Print the largest of x, y c) Print the largest of x, y ,z. Assume variables are all ints or floats.
- Blocks are signalled only by indentation.

```
#Program a)
x=19
if x>9:
    print(x)
    print('then block')
print('End of program')
```

```
#Program b)
x,y=8,4
if x>y:
    print(x)
    print('then block')
else:
    print(y)
    print('else block')
print('End of program')
```

```
#Program c)
x,y,z=4,3,5
if x>y and x>z:
    print(x)
    print('then block')
elif y>z:
    print(y)
    print('elif block')
else:
    print(z)
    print('else block')
print('End of program')
```

Indentation has semantic meaning

- Indentation is used in Python for demarcating blocks. In languages like C, C++ and Java this is done using the open and close curly braces. The use of indentation for blocks means the program is automatically formatted for readability.
- All consecutive lines of code with the same indentation form a block.

Strings

- A string is a non scalar value and has indvidual characters as components.
- String values can be written as a sequence of characters inside single or double quotes. For example,
  'This is a string'
  "This is also a string"
  'string with \' single quote'
  "string with ' single quote"
  'string with " double quote'
  '' and "" represent the empty string

Common operations on strings

Strings have almost 40 defined operations. We look at a few useful ones. Let `s='hello there'` be a string.

- `len(s)` is 11 the length of s i.e. number of characters in s.
- `s.index('t')` is 6 position of 't' in s - index starts from 0.
- `s.count('e')` is 3 - number of 'e' in s.
- `s[3:8]` is 'lo th' - substring from 3 till 8 - includes 7 not 8.
  `s[4]` is 'o' - the character at index 4.
  `s[-2]` is 'r' - second character from right.
- The slice operation is very general [start:stop:step]. Default step is 1, if start, stop are missing defaults to start and end of string. s[:7,2] will give 'hlot'.
- `'hello'+' '+'there'` will give 'hello there'. '+' acts as concatenation for strings. It is overloaded.
- `3*'hello'` is 'hellohellohello'. '*' acts as a repeater.

Reading data from terminal - input command

- Many programs need input from the user. At run time this can be given by typing in the input (shows up on the terminal) or by specifying files from which it will be read.
- The command `input(Prompt string)` prints the Prompt string and returns any typed-in input (terminated by a newline - Enter key) as a string.
- The string read in can be typecast to other types if the string can be legally converted to that type. For example, int('87690') will convert the string to the integer 87690. If conversion is not possible an error will be signalled.

Program using strings

Program to print the reverse of an integer where the reversed integer can contain leading 0s.

Program using strings

Program to print the reverse of an integer where the reversed integer can contain leading 0s.

```
stri=input('Enter an integer:') # stri will be a string
i=int(stri) #converted to integer
if i==0:
    revi='0'
elif i<0:
    revi='-'+(stri[1:])[::-1]
else:
    revi=stri[::-1]
print('Reversed',revi)
# Notice how both the string and integer representations
# are needed.
```

Iteration - while loop

Program to print the number of digits in an integer - no leading 0s.

Iteration - while loop

Program to print the number of digits in an integer - no leading 0s.

```
i=int (input('Enter an integer:'))
if i<0:
    i=-i
noOfDigits=0
if i==0:
    noOfDigits=1
else:
    while i>0:
        noOfDigits+=1
        i=i//10
print('No of digits=',noOfDigits)
```

Iteration - while, another example

Program to find the largest +ve integer that divides both +ve integers x, y. For example if x=26, y=91 than answer is 13. The largest divisor is called hcf or gcd.

Iteration - while, another example

Program to find the largest +ve integer that divides both +ve integers x, y. For example if x=26, y=91 than answer is 13. The largest divisor is called hcf or gcd.

```
x=int(input('Enter first +ve integer='))
y=int(input('Enter second +ve integer='))
hcf=1
if x>1 and y>1: # this is the interesting case
    while (x!=y):
        if x>y:
            x=x-y
        else:
            y=y-x
    hcf=x
elif x<=0 or y<=0:
    hcf='Illegal input'
else:
    hcf= 1
print('HCF=',hcf)
```

Iteration - for loop

The `for` loop is normally used when we know the number of times a loop will be executed.
Program to add numbers from 1 to n, where n is read in.

Iteration - for loop

The `for` loop is normally used when we know the number of times a loop will be executed.
Program to add numbers from 1 to n, where n is read in.

```
n=int(input('Enter +ve integer='))
sum=0
if n>=0:
    #upto but not including second argument of range
    for i in range(1,n+1):
        sum+=i
    print('Sum=',sum)
else:
    print('Illegal input')
```