Python:- created by Guido van Rossum

It’s a general purpose interpreted, interactive, object oriented and high level programming language. It has few keywords, simple structure and a clearly defined syntax which allows the student to pick up the language quickly

* High level programming language
* General purpose
* Interpreted
* Interactive
* Object oriented
* Frequently uses English keywords instead of punctuation
* Fewer syntactical constructions
* Beginner language

Features:-

* Easy to learn:- (few keywords, simple structure, clear defined syntax)
* Easy to read
* Easy to maintain
* A broad standard library
* Interactive mode
* Portable
* Extendable
* Databases
* GUI Programming
* Scalable
* Can be used as scripting language or can be compiled to byte code
* Provides high level dynamic data types and supports dynamic type checking
* Automatic garbage collection
* Easily integrated with C, C++, COM, CORBA, ActiveX, Java.

Python Identifiers:-

* To identify a variable
* Starts with A to Z or a to z or (\_) followed by digits and alphabets
* Does not allow punctuation characters (@, $, %)
* Case sensitive
* Class start with an uppercase letter and other start with lowercase letter.
* Starting with \_ means its private
* Starting with \_\_ indicates strongly private
* Starts and ends with \_\_ indicates identifier is language defined special name

Lines & Identation:-

* Does not provide braces for block indication
* Line indentation, rigidly enforced
* Allows the use of line continuation character (\) for continuation
* Statements within [], {}, () do not need to use (\)

Quotation:-

* Accepts (‘), (“ “), (‘’’ ‘’’ or “”” “””) quotes
* Word = ‘word’
* Sentence = “sentence”
* Paragraph = “”” This is paragraph”””

Multi Line Assignment:- semicolon (;) allows multiple statements on the single line. It allows multiple assignment.

Header lines begins with keyword & terminate with colon ( : )

Standard data types:

1) Numbers

2) String

3) List

4) Tuple

5) Dictionary

It supports 4 numerical types:-

* Int
* Long (long, octal, hexadecimal)
* Float
* complex

# Day 2:-

**Python Data Types:-**

**1. Python Numbers:-**

It supports 4 numerical types:-

* Int (50)
* Long (long, octal, hexadecimal) (518245262L)
* Float (125.0)
* Complex (3.14j)

**2. Strings:-**

* Continuous set of characters represented in quotation marks
* Subset can be taken using slice operator ([], [:])

e.g. welcomeStr = “Welcome to world of Python”

**3. List:-**

* Most versatile pyhon compound data type
* Similar to arrays in ruby
* All elements can be of different data type
* Can be accesses using slice operator
* Enclosed with square brackets

list = [‘abcd’, 786, 2.23, ‘john’, 70.2]

print list[1:3] # print elements starting fron 2nd to 3rd

Ans: 786, 2.23

**4. Tuples:-**

* Similar to list
* Enclosed within parentheses ( () ) unlike list ( [])
* Cannot be updated
* Read only lists

tuple = (‘abcd’, 786, 2.23, ‘john’, 70.2)

list = [‘abcd’, 786, 2.23, ‘john’, 70.2]

tuple[2] = 1000 # invalid with tuple, updating is not allowed within tuple

list[2] = 1000 # it will work fine, modification is allowed in list

**5. Dictionary:-**

* Hash table type
* Enclosed by curly braces
* Represented by keys and value pairs

dict = {‘name’: ‘Anjali’, ‘code’: 1309, ‘dept’: ‘search in progress’}

**dict.keys() & dict.values()**

dictionary keys is ['dept', 'code', 'name'] dictionary values is ['search in progress', 1309, 'Anjali']

**Data Type Conversion:-**

* Performs conversions between the built in types.
* Use the type name as functions

e.g. int(x[,base]), float(x), str(x), eval(str), tuple(s), long(x[,base]), complex(real[,imag]), repr(x), list(s), set(s), dict(d), frozenset(s), chr(x), unichr(x), ord(x), hex(x), oct(x)

Day 4:-

Python Operators:-

1. Arithmetic Operator
2. Comparison Operator
3. Assignment Operator
4. Logical Operator
5. Bitwise Operator
6. Membership Operator
7. Identity Operator

**Arithmetic Operator:** (+, -, \*, /, %, \*\*, //)

**Comparison Operator:** (==, !=, <>, >, <, >=, <=)

**Assignment Operator:**

|  |  |
| --- | --- |
| = | Assignment |
| += | Add AND |
| -= | Subtract AND |
| \*= | Multiply AND |
| /= | Divide AND |
| %= | Modulus AND |
| \*\*= | Exponent AND |
| //= | Floor Division |

**Bitwise Operator:-**

|  |  |
| --- | --- |
| & | AND |
| | | OR |
| ^ | XOR |
| ~ | Ones complement |
| << | Left Shift |
| >> | Right Shift |

**Logical Operator:- (**and, or, not)

**Membership Operators:-**

* Test for membership in a sequence such as strings, lists or tuples
* **in** 🡪 evaluates to true if it finds
* **not in** 🡪 evaluates to true if it does not finds

**Identity Operator:**

* compare the memory location of two objects
* **is** 🡪 evaluates to true if the variables on either side of the same object
* **is not 🡪** evaluates to false if the variable on either side of the operator point to same object

**Day 5:-**

**Decision Making Conditions**:-

* anticipation of conditions occurring while execution of program and specifying action according to the conditions
* if, if else, if elif else
* assumes any nonzero and non null values as TRUE
* either zero or null assumed as false

**e.g.** if (var == 200) : print “value of expression is 100”

**Loops:-**

* allows to execute a statement or group of statement multiple times.
* While loop – repeats the loop if true
* For loop – execute multiple times and abbreviates the code
* Do while – execute minimum one time and repeats the loop till condition is true

**Loop Control Statements:-**

* Changes execution from its normal sequence.
* When execution leaves a scope, all automatic objects that were created in that scope are destroyed
* **break:-** terminates the loop and transfer execution to the statement immediately following the loop
* **continue:**- causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
* **pass:**-when a statement is required syntactically but you do not want any command or code to execute.

**Mathematical Functions:-**

* functions that perform mathematical calculations
* abs(x), ceil(x), cmp(x,y), exp(x), fabs(x), floor(x), log(x), log10(x), max(x1, x2….), min(x1, x2….), modf(x), pow(x,y), round(x [,n]), sqrt(x)

**Random number functions:-**

* used for games, simulations, testing, security and privacy applications.
* Choice(seq), randrange([start,] stop [,step]), random(), seed([x]), shuffle(lst), uniform(x, y)

**Trigonometric Functions:-**

* Functions that perform trigonometric calculations
* acos(x), asin(x), atan(x), atan2(y,x), cos(x), hypot(x,y), sin(x), tan(x), degrees(x), radians(x)

**Strings:-**

* treats single quotes the same as double quotes
* to access the substring use the square brackets for slicing along with the index
* **var1 =** ‘Hello World’

**print** “var1[0]: “, var1[0] # var1[0]: H

* update and existing string by assigning a variable to another string

print “Updare String :- , var[:6] + ‘Python’ # Ans Hello Python

* string format operator % is unique to strings and makes up for the pack of having functions (from C’s printf() family
* print “My name is %s and weight is %d” (“%(‘Zara, 21)
* complete set of symbols used along with %

#Day 6:-

Set of symbols which can be used along with %

|  |  |
| --- | --- |
| %c | Character |
| %s | String |
| %i | Signed decimal integer |
| %d | Signed decimal integer |
| %u | Unsigned decimal integer |
| %o | Octal integer |
| %x | Hexadecimal integer (lowercase letter) |
| %X | Hexadecimal integer (Uppercase letters) |
| %e | Exponential notation (lowercase e) |
| %E | Exponential notation (Uppercase E) |
| %f | Floating point real number |
| %g | Shorter of %f and %e |
| %G | Shorter of %f and %E |

**Triple Quotes:-**

* allowing strings to span multiple lines, including verbatim NEWLINES, TABs and any other special characters
* consists of three consecutive single or double quotes.

parastr = “”” this is a long string. It contains

Tab( \t). New lines explicitly given like

This within the brackets [\n] this is new line“””

Ans:- this is a long string. It contains

Tab( ). New lines explicitly given like

This within the brackets

this is new line

* every single special character converted to its printed form
* NEWLINES occur either with explicit carriage return at the end of a line or its escape code.
* Raw strings do not treat backslash as a special character

print ‘C:\\nowhere’ # C:\nowhere

with r’expression’

print r’C:\\nowhere # C:\\nowhere

**Unicode String:-**

* Strings stored internally as 8 bit ASCII
* Unicode strings are stored as 16-bit Unicode

**Python sequence types:-**

* 6 built in types of sequence
* Str
* Unicode Strings
* Lists
* Tuples
* Buffers
* Xrange Objects

**Buffers :-**

* Not directly supported by Python syntax
* Can be created by calling built in function buffer()
* Don’t support concatenation or repetition

**Xrange Objects:-**

* Similar to buffers no specific syntax
* Can be created using xrange function
* Don’t support slicing, concatenation or repetition
* Using in, not in, min() or max() on them is inefficient

**Lists:-**

* Most basic data structure is sequence.
* Each element assigned number i.e. position or index 1st -> 0, 2nd -> 1 and so on.
* Common ones are list and tuples
* Common operations:- indexing, slicing, adding, multiplying, checking for membership
* List of comma-separated values(items) between square brackets
* Items need not be of same type.

Deleting element in list:-

list1 = [“ab”, “b”, “c”, “d”]

del list[2]

* List respond to + and \* operator, they mean concatenation, repetition here also
* Indexing and slicing work the same way for lists as they do for strings

**Built in List functions & methods:-**

|  |  |
| --- | --- |
| Cmp(list1, list2) | For comparison |
| Len(list) | Total length |
| Max(list) | Max value is returned from list |
| Min(list) | Min value is returned from list |
| List(seq) | Converts a tuple into list |

* list.append(obj), list.count(obj), list.extend(seq), list.index(obj), list.insert(index, obj), list.pop(obj = list[-1]), list.remove(obj), list.reverse(), list.sort([func])

**Tuples:-**

* comma-separated values between parentheses also
* tup1 = (1, 2, 3, 4, 5)
* tup2 = (); # empty tuple
* tup3 = (50, ); # for single element comma need to be specified
* like string, tuple can be slice, concatenated and so on.
* Square brackets for slicing along index or indices to obtain value available at that index.
* print “tup1[0]: “, tup1[0];

**Delete tuple:-**

* removing individual tuple elements is not possible
* remove an entire tuple, just use the del statement
* del tup;

\* any set of multiple objects, comma-separated, written without identifying symbols i.e. brackets for lists, parentheses for tuple etc, default to tuples.

**# Day 7:-**

**Built in Tuple Func:-**

|  |  |
| --- | --- |
| cmp(tuple1, tuple2) | Compares elements of both tuple |
| len(tuple) | Length of tuple |
| max(tuple) | Return max value from tuple |
| min(tuple) | Return min value from tuple |
| tuple(seq) | Converts a list into tuple |

**Dictionary:-**

* key is separated from its value by a colon( : )
* items are separated by commas
* enclosed in curly braces like {}
* keys are unique while values may not

dict = {‘Name’: ‘zoo’, ‘age’ : 7, ‘class’: ‘first’}

del dict[‘Name’] # Remove entry with key ‘Name’

dict.clear(); # remove all entries in dict

del dict; # delete entire dictionary

* value of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers or types.

dict = {[‘Name’] : ‘zero’, ‘age’ : 7 }

print “dict[‘Name’]: “, dict[‘Name’] # list is not allowed as key

* similar to hashes in ruby.

**Python Date & Time:-**

* can handle date and time in several ways
* time and calendar modules help track dates & times
* time are expressed in seconds since 12:00 am Jan 1, 1970(epoch)
* time modules provides functions for working with times and for converting between representations.
* Dates before the epoch cannot be represented in this form
* Dates in the for future also cannot be requested this way.
* Tuple is equivalent to struct\_time structure

**Struct\_time structure**

|  |  |  |
| --- | --- | --- |
| Index | Attributes | Values |
| 0 | tm\_year | 2008 |
| 1 | tm\_mon | 1 to 12 |
| 2 | tm\_mday | 1 to 31 |
| 3 | tm\_hour | 0 to 23 |
| 4 | tm\_min | 0 to 59 |
| 5 | tm\_sec | 0 t 61(60 or 61 are leap sec) |
| 6 | tm\_wday | 0 to 6 |
| 7 | tm\_yday | 1 to 366(Julian Day) |
| 8 | tm\_isdst | -1, 0, 1, -1 means library determines DST |

* **Getting current time:-**

localtime = time.localtime(time.time())

* Can format any time as per your requirement

localtime = time.asctime(time.localtime(time.time()))

* Wide range of methods to play with with yearly and monthly calendar

# for printing calendar

import calendar;

cal = calendar.month(2018, 1)

print cal

**time module:-**

* Provides various functions for working with times and for converting between representations

**Calendar module:-**

* Supplies calendar-related functions,
* Functions to print a text calendar for a given month or year

**Other modules:-**

* DateTime Modules
* Pytz modules
* DateUtil Modules

# Day 8:-

**Functions:-**

* Block of organized, reusable code for specific action
* Used for better modularity for application
* High degree of code reusing

**Syntax:-**

def functionname (parameter):

“function 1”

function\_suite

return [expression]

**Example:**

def printme (str):

print str

return;

**Calling a function:-**

printme(“Calling the print me function”)

**Pass by reference:-**

* Parameters passed by reference
* Changes done inside method will reflect outside

e.g.

#!/usr/bin/python

def changeme( mylist ):

mylist.append([1, 2, 3, 4]);

print "pass by reference my list inside function is ", mylist # Ans: [10, 20, 30, 1, 2, 3, 4]

return

mylist = [10, 20, 30]

changeme( mylist );

print "pass by reference mylist outside function is ", mylist # Ans: [10, 20, 30, 1, 2, 3, 4]

**Pass by value:-**

* Variable scope is limited inside method only
* an example of pass by value changes made inside method will not reflect outside

def changeme2( mylist ):

mylist = ([1, 2, 3, 4]);

print "pass by value my list inside function is ", mylist # Ans: [1, 2, 3, 4]

return

mylist = [10, 20, 30]

changeme2( mylist );

print "pass by value mylist outside function is ", mylist # Ans: [10, 20, 30]

**Function Arguments Types:-**

1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable length arguments

**Required arguments:-**

* Arguments to be passed in correct positional order

# a basic function

print "Welcome to Python Function argument type examples"

def printStr(str):

print "str is: ", str # it will print the string

return;

printStr("This is an example for a basic function")

print "\n\n"

**Keyword Argument function:-**

* Keyword to be declared while calling functions
* Caller identifies the arguments by parameter name
* Arguments can be placed out of order

# an example of keyword arguments function

print "This is an example for a keyword argument function"

def printinfo(name, age):

print "name is: ", name, " and age is: ", age

return;

printinfo("Anjali", "26")

printinfo(age = "26", name = "Anjali")

print "\n\n"

**Default arguments:-**

* Does not require argument to be passed for function calling
* If argument is not passed, default value is taken

print "This is an example for a default argument function"

def printinfo(name, age = "35"):

print "name is ", name, "age is ", age

return;

printinfo("Neha", "26")

printinfo("Sameeksha")

print "\n\n"

**Variable length arguments:-**

* Used when no of arguments is not fixed
* More requirements is required then specified
* (\*) is placed before variable name for assignment of variable arguments
* Arguments remains empty if no additional arguments specified

# variable length arguments function

print "This is an example for a variable length argument function"

def variableLengthArguments (arg1, \*vartuple):

print "arg1 is ", arg1

for var in vartuple:

print var

return;

variableLengthArguments (10)

variableLengthArguments (70, 60, 50)

print "\n\n"

**Anonymous Function:-**

* Functions which are not declared in standard manner
* Uses lambda keyword to create small anonymous functions
* Lambda forms can take any no of arguments
* Returns just one value in form of an expression
* Lambda cannot contain commands or multiple expressions
* It cannot be a direct call because lambda requires an expression
* Lambda have own local namespace
* Lambda’s one line version of a function
* Lambda purpose if by passing function stack allocation during invocation for performance issues

**Syntax:**

lambda [arg1, [arg2, ….. argn]] : expression

# Anonymous functions(lambda):- functions not defined using def keyword

print "This is an example for a lambda(Anonymous function)"

lambdaSum = lambda arg1, arg2 : arg1 + arg2

print "value of total is ", lambdaSum( 10, 20 )