Anodiam

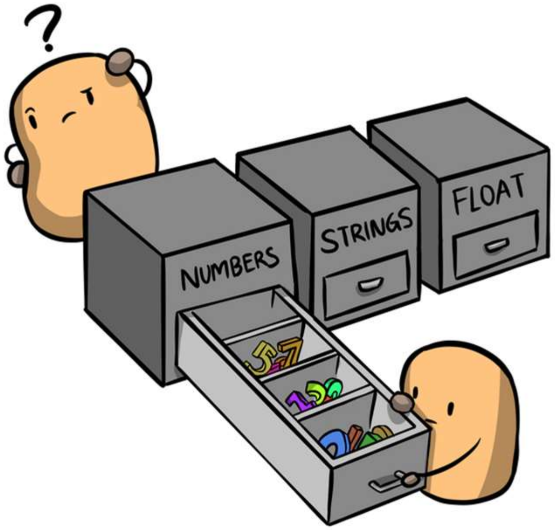
Education that enlightens!

PYTHON PROGRAMMING: INTRODUCTION & INSTALLATION

- **General Purpose Language:**
  - Machine Learning AI
  - GUI, Web, Software Dev
- **Python Features:**
  - Interpreted
  - Object oriented
  - High Level
  - Current version 3.x
- **Download Install & Run:**
  - https://www.python.org
  - Start > Python IDLE
  - Write a line of code

    e.g. 2+3 OR print(“Hello World”)
- **Community version of PyCharm IDE:**
  - https://www.jetbrains.com/pycharm/download
  - Install 64 bit (depending on system)
  - Start > PyCharm > Create New Project > filename.py
  - print(“Hello World”)
  - Run the file

PYTHON PROGRAMMING: VARIABLES & STRINGS



❖ Types:

- **int** e.g. 4, 1000, -34 etc.
- **float** e.g. 33.4, -0.0005
- **string** e.g. ‘My friend Tom’, ‘cricket’ etc.

❖ Assignment:

- x = 2
- name = ‘Rupa’

❖ Strings:

A	n	o	d	i	a	m		R	o	c	k	s	
0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	→**index**
-13	**-12**	**-11**	**-10**	**-9**	**-8**	**-7**	**-6**	**-5**	**-4**	**-3**	**-2**	**-1**	

Strings are immutable:

```
str1 = “My Name”
str1[3] = ‘n’        # Error
```

Anodiam

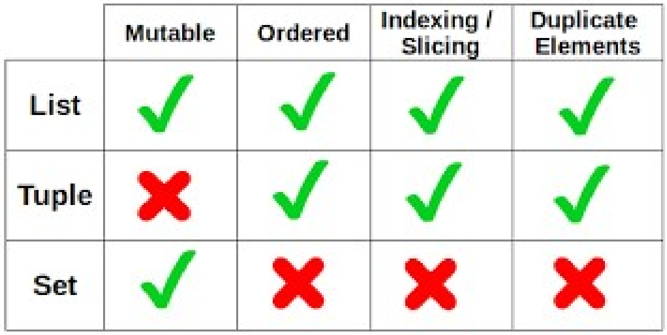
Education that enlightens!

PYTHON PROGRAMMING: LISTS, TUPLES & SETS

- **Lists: [ ]**
  - Mutable
  - E.g.: myList = [23, 91, 9, 507]
  - Functions used in List:
    - append()
    - insert()
    - remove()
    - pop()
    - clear()
    - extend()
    - max(list)
    - min(list))
    - sum(list))
    - sort()
    - sort(reverse=True)

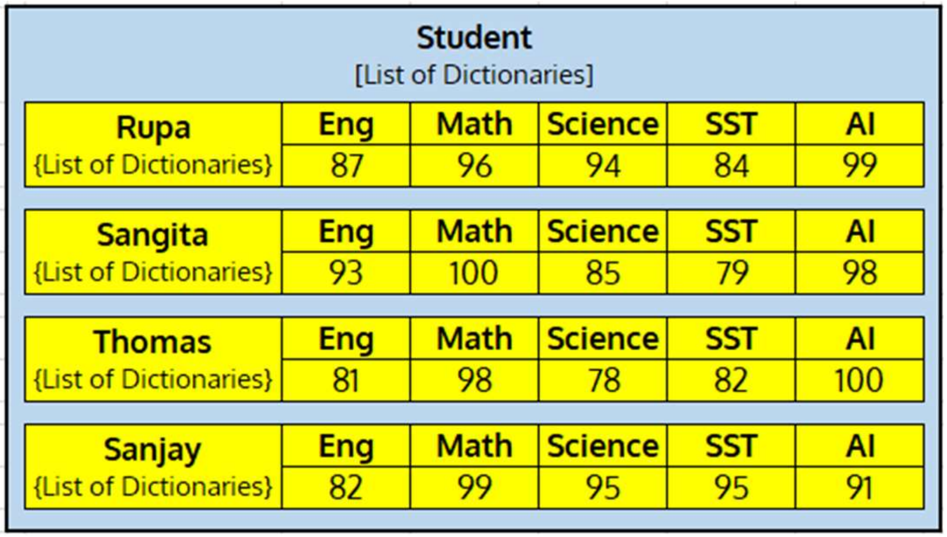
- **Tuples: ( )**
  - Immutable
  - Iteration is faster than list
  - E.g.: myTuple = (23, 91, 9, 507)

- **Sets: { }**
  - Like mathematical sets: do not consider repetition
  - E.g.: mySet = {23, 91, 9, 507, 91}
  - Uses hashing for performance
  - No index for sets
  - Mutable
  - Functions used in Set:
    - add()
    - remove()
    - set1.update(set2)



	Mutable	Ordered	Indexing / Slicing	Duplicate Elements
List	✓	✓	✓	✓
Tuple	✗	✓	✓	✓
Set	✓	✗	✗	✗

PYTHON PROGRAMMING: DICTIONARIES



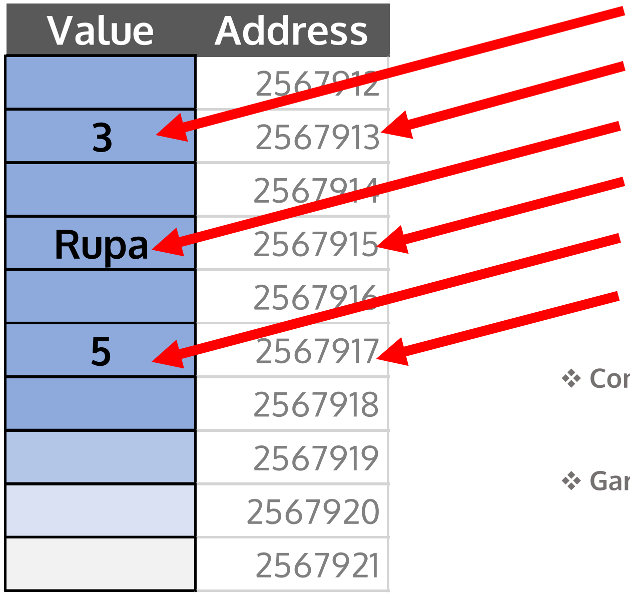
Student
[List of Dictionaries]

Rupa {List of Dictionaries}	**Eng**	**Math**	**Science**	**SST**	**AI**
	87	96	94	84	99
Sangita {List of Dictionaries}	**Eng**	**Math**	**Science**	**SST**	**AI**
	93	100	85	79	98
Thomas {List of Dictionaries}	**Eng**	**Math**	**Science**	**SST**	**AI**
	81	98	78	82	100
Sanjay {List of Dictionaries}	**Eng**	**Math**	**Science**	**SST**	**AI**
	82	99	95	95	91

- **Dictionaries: { key : value }**
  - ❑ Key must be unique and immutable
  - ❑ Key can ne numeric or non-numeric
  - ❑ Access values through key/get( ) function
  - ❑ Can zip 2 lists (key-list & value-list) to form a dictionary
  - ❑ Real Life dictionaries can have nested lists, tuples, sets & other dictionaries inside

Education that enlightens!

PYTHON PROGRAMMING: VARIABLES



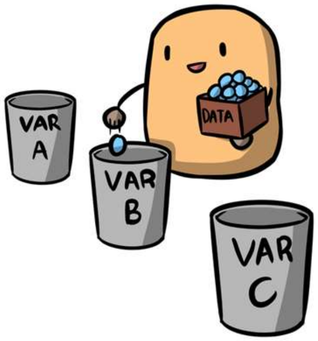
Value	Address
	2567912
3	2567913
	2567914
Rupa	2567915
	2567916
5	2567917
	2567918
	2567919
	2567920
	2567921

- **Variables Trivia:**
  - ☐ num = 3
  - ☐ id(num)
  - ☐ name = 'Rupa'
  - ☐ id(name)
  - ☐ Two variables assigned same values: a = 5, b=5
  - ☐ id(a) will therefore point to the same memory as id(b)
- **Constants:**

  Actually variables with names in CAPS e.g. PI = 3.14
- **Garbage Collection:**

  Unused objects and variables are garbage collected by Python

PYTHON PROGRAMMING: DATA TYPES & OPERATORS



❖ Data Types:

- **None:** var not assigned (null)
- **Numeric:**
  - **Int**
  - **Float**
  - **Complex** e.g. 6+4.5j
  - **Bool** True, False

Sequence

- **List**
- **Tuple**
- **Set**
- **String**
- **Range**
- **Dictionary**

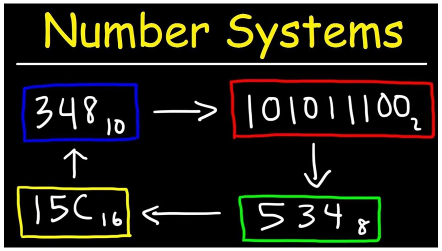
** Character data type absent in Python

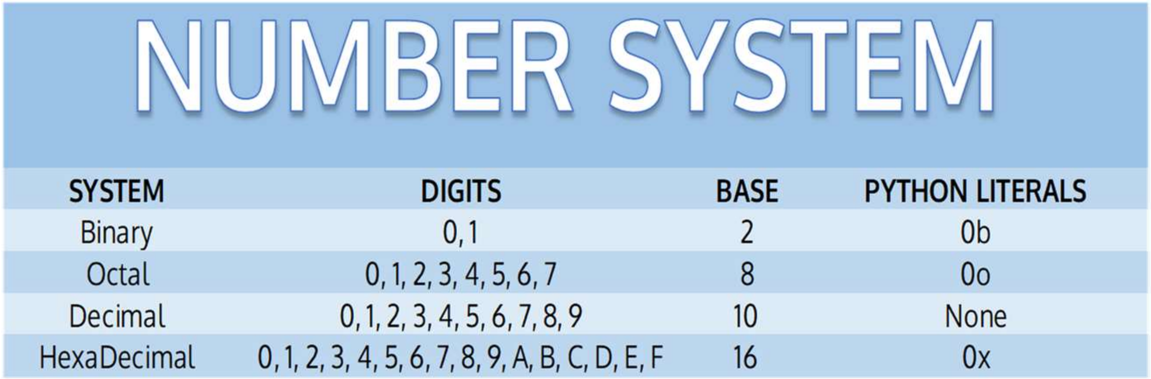
❖ Types of Operators :

Arithmetic	e.g. + - * / % etc.
Assignment	assigning values (=)
Relational	compare two variables (">" "<" "==" ">=" "<=" "!=")
Logical	"AND" "OR" "NOT"
Unary	negating a value
Bitwise	"XOR" "<<" ">>" etc.

Anodiam
Education that enlightens!

PYTHON PROGRAMMING: NUMBER SYSTEM


Number Systems
348_{10}
101011100_2
534_8
$15C_{16}$



NUMBER SYSTEM

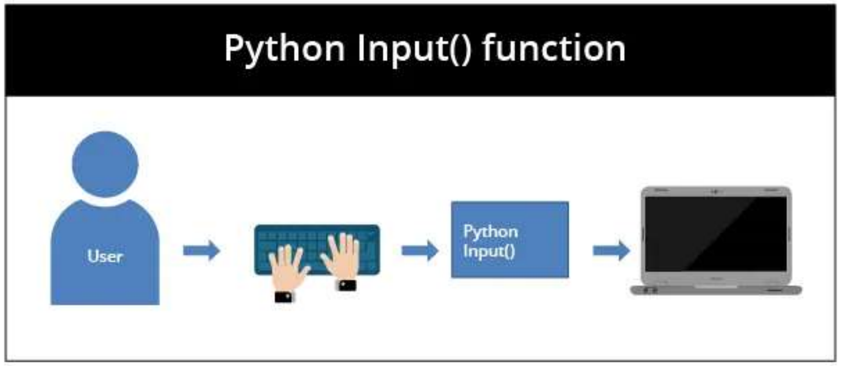
SYSTEM	DIGITS	BASE	PYTHON LITERALS
Binary	0, 1	2	0b
Octal	0, 1, 2, 3, 4, 5, 6, 7	8	0o
Decimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9	10	None
HexaDecimal	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F	16	0x

Your MAC Address is:
4C:7C:5F:0B:54:05

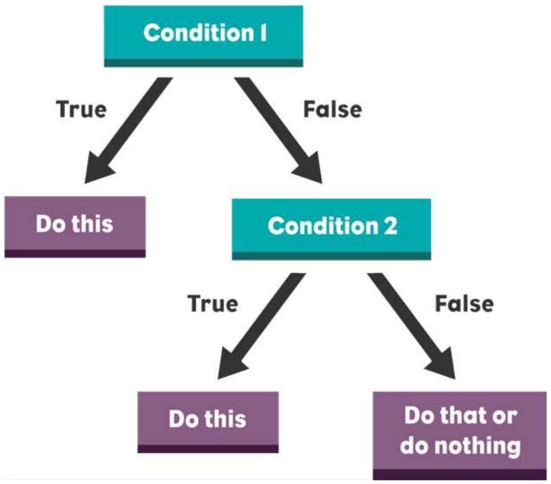
❖ **print(num)**	Decimal to Binary
❖ **print(bin(num))**	Binary to Decimal
❖ **print(oct(num))**	Decimal to Octal
❖ **print(hex(num))**	Decimal to HexaDecimal

PYTHON PROGRAMMING: USER INPUT

- **input():** Delays the execution of program and asks user for input
    - ❑ Strings, Integers, Floats, Characters and Expressions can be asked from the user by passing proper functions inside input()
- **eval():** Evaluates any mathematical expression


Python Input() function
User
Python Input()

PYTHON PROGRAMMING: IF ELSE STATEMENT


Condition 1
True
False
Do this
Condition 2
True
False
Do this
Do that or do nothing

- **If Statement:** Uses if keyword followed by condition
- **If-Else Statement:** Used to execute both the true part and the false part of a condition. If true, the if block code is executed. If false, the else block code is executed
- **Nested If:** One or more statements present inside another if statement. Used when a variable has to be processed multiple times
- **If-Elif-Else:** It checks the if statement condition. If false, the elif statement is executed. If elif is false, the else statement is executed

PYTHON PROGRAMMING: FOR & WHILE LOOPS

- **For:** Definite iteration. Iterates predefined number of times
- **While:** Indefinite Iteration. Keeps iterating until the condition is false
- **The Else Clause:** Python exclusive
  - **For Else**
  - **While Else**
- **Loop Control Statements:**
  - **Break –** Terminates the loop
  - **Continue –** Forces to execute the next iteration of loop
  - **Pass –** Null operation

PYTHON PROGRAMMING: ARRAY

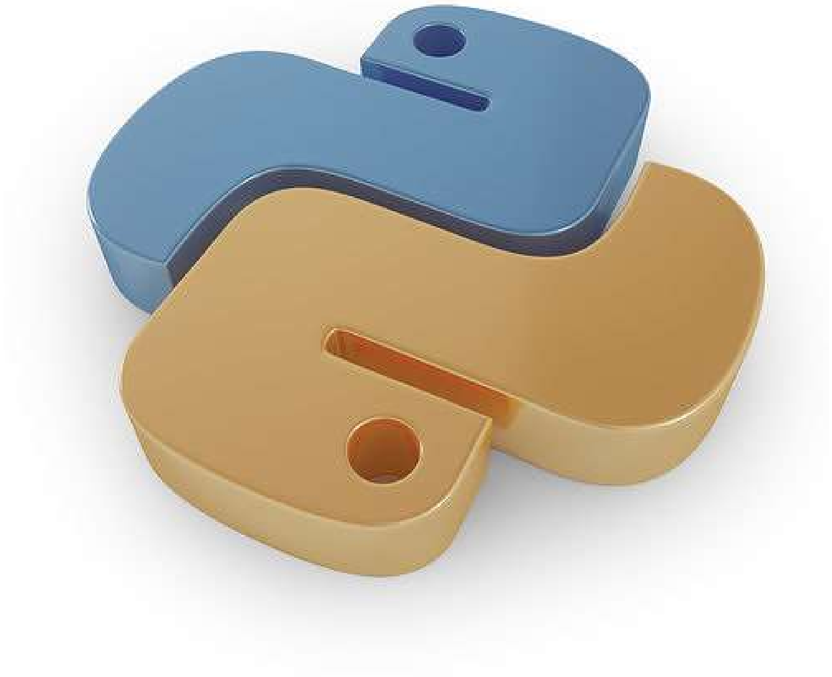
TYPECODE	CHARACTER TYPE	PYTHON TYPE	MINIMUM SIZE IN BYTES
b	Signed Char	int	1
B	Unsigned Char	int	1
u	Py_UNICODE	unicode character	2
h	Signed Short	int	2
H	Unsigned Short	int	2
i	Signed int	int	2
I	Unsigned int	int	2
l	Signed Long	int	4
L	Unsigned Long	int	4
f	Float	float	4
d	Double	float	8

- **Array:**
  - Array can be one dimensional or multi dimensional
  - Import array module to work with arrays
  - All elements of an array should be of same type
  - Can take inputs from user to form an array
- **Functions:**
  - typecode()
  - buffer_info()
  - array()
  - extend()
  - append()
  - pop()

PYTHON PROGRAMMING: NUMPY

- **Numpy:**
    - general purpose array-processing package
    - Used mainly for multidimensional arrays
    - Efficient even with 1D array. For e.g.- mentioning typecode is not necessary
- **Array Creation:**
    - array() – array with given data
    - linspace() – array of range divided in specified number of parts
    - logspace() – array of range divided in parts but log values
    - arange() – like linspace but breaks the range in a progressive series
    - Zeros() – array of 0's
    - Ones() – array of 1's

PYTHON PROGRAMMING: MORE ON ARRAY



- **Array Addition:**
  - ❑ Any variable can be added with each element
  - ❑ **sum():** Sum of all the array elements
  - ❑ Arrays support various types of mathematical operations
- **Copying Array:**
  - ❑ **Aliasing:** Assigning an existing array to a new one stored at same address
  - ❑ **Shallow Copy:** view() function copies the array to form a new but dependent one at different address
  - ❑ **Deep Copy:** copy() function does the sane but the arrays stay independent

PYTHON PROGRAMMING: MATRICES

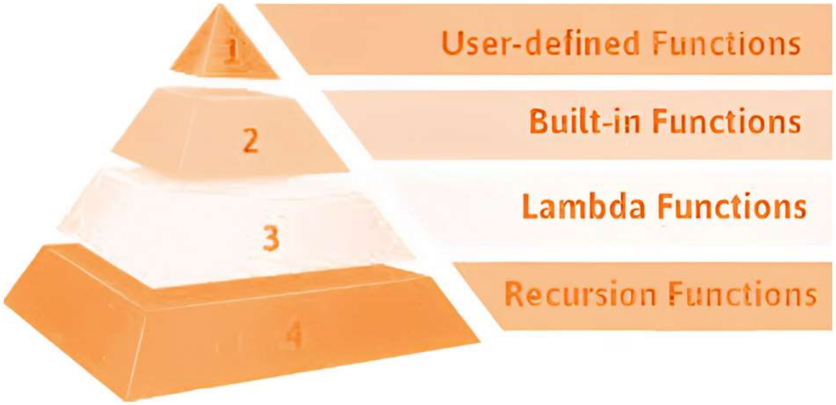
- **Array Addition:**
  - ☐ Any variable can be added with each element
  - ☐ **sum():** Sum of all the array elements
  - ☐ **Vectorized Operation:** Adds elements of two arrays to form a new array

Education that enlightens!

PYTHON PROGRAMMING: VECTORIZED OPERATION

- **Array Addition:**
  - ☐ Any variable can be added with each element
  - ☐ **sum():** Sum of all the array elements
  - ☐ **Vectorized Operation:** Adds elements of two arrays to form a new array

PYTHON PROGRAMMING: FUNCTIONS

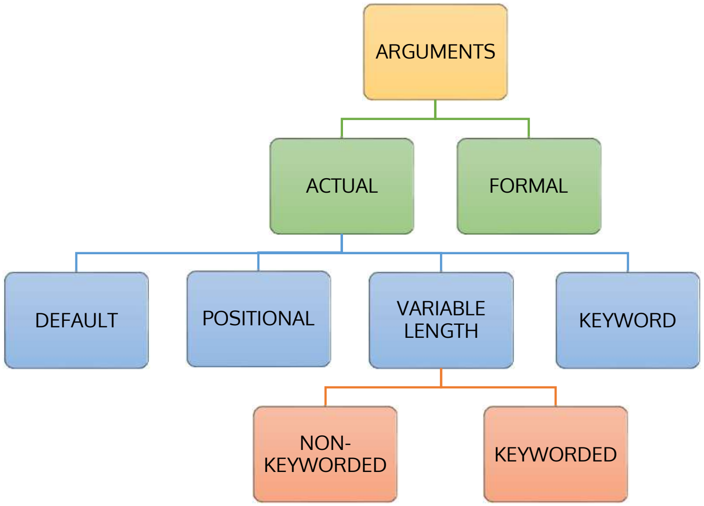

1
User-defined Functions
2
Built-in Functions
3
Lambda Functions
4
Recursion Functions

- **Function** is a block of code which only runs when called
- **Creating New Function:**
  - **def** keyword creates a new function
  - Can assign many functions/statements
  - Can be called anywhere and anytime
  - Saves time and space in coding
  - Nested functions
- **Return Statement:** Used inside a function or method to send the result back to caller
- **Parameters** are variables inside the parentheses in a defined function. e.g. fn(x, y)
- **Arguments** are values sent to the function when it is called. e.g. fn(2, 3)

****Python's argument passing is neither "pass by value" nor "pass by reference" but "pass by object reference"**

Education that enlightens!

PYTHON PROGRAMMING: ARGUMENTS


ARGUMENTS
ACTUAL
FORMAL
DEFAULT
POSITIONAL
VARIABLE LENGTH
KEYWORD
NON-KEYWORDED
KEYWORDED

- **Formal** arguments are parameters passed while defining a function
- **Actual** arguments are the values passed to the function when called
  - ☐ **Default** arguments are the values assigned while defining a function
  - ☐ **Positional** arguments must be specified in the proper order
  - ☐ **Keyworded** arguments use keys to assign values even if they are not sorted in order
  - ☐ **Variable Length** arguments are used when the number of values is unknown

    e.g. **input()** where user might give multiple values

    ***args** used for passing a variable number of arguments to a function

    ****kwargs** used for the same as *args but with keys assigned to each of them

PYTHON PROGRAMMING: GLOBAL KEYWORD & RECURSION

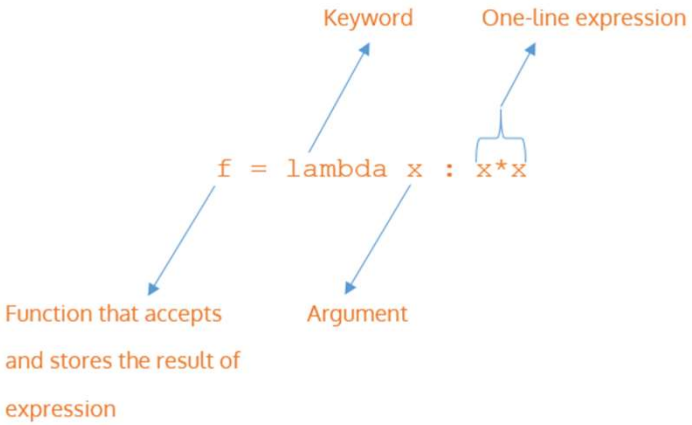
GLOBAL
LOCAL

- **Global** variables have a global scope and assigned outside functions
- **Local** variables have a local scope only inside the function they are assigned to
- **Global Keyword** lets access and modify the global variable even inside a function
- **globals():** function returns the dictionary of current global symbol table
  - ❑ Changes a global variable without affecting local ones if called inside a function
- **Recursion** is a process where a function calls itself

  e.g. Program to find factorial of a number
  - ❑ **sys.getrecursionlimit():** default limit of recursion is 1000
  - ❑ **sys.setrecursionlimt(n):** changes the limit of recursion to desired value “n”

Education that enlightens!

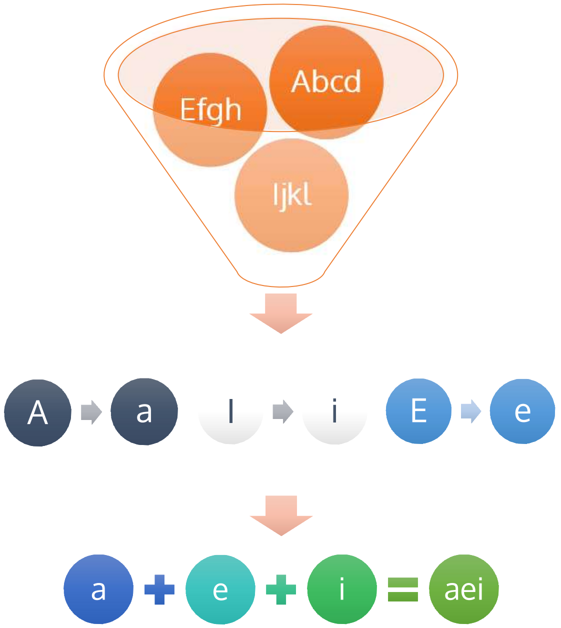
PYTHON PROGRAMMING: ANONYMOUS FUNCTIONS


Keyword
One-line expression
f = lambda x : x*x
Function that accepts and stores the result of expression
Argument

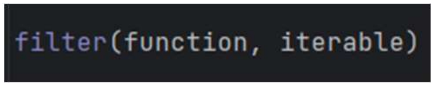
- **Lambda** aka anonymous functions are functions without names
    - Used when a small, one-time-use function is required
    - Initiated with the keyword **“lambda”** instead of “def”
- **Limitations:**
    - Limited to a single expression.
    - They are often less readable than named functions
    - No proper names or docstrings make it harder to understand their purpose

Anodiam
Education that enlightens!

PYTHON PROGRAMMING: FILTER MAP & REDUCE

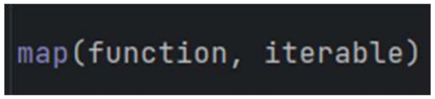


❖ **filter()** is an in built function used for fetching elements from a collection (e.g. list or array) based on a set condition



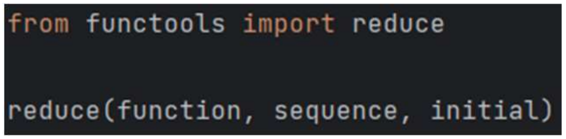
```
filter(function, iterable)
```

❖ **map()** function transforms each element in a collection using a given function by applying it to each element



```
map(function, iterable)
```

❖ **reduce()** aggregates elements in a collection to produce a single result by repeatedly applying a binary function to them



```
from functools import reduce

reduce(function, sequence, initial)
```

**** initial can be assigned with a value to be aggregated with the passed sequence**

Education that enlightens!

PYTHON PROGRAMMING: DECORATORS

- **Decorators** can modify functions (or methods) and classes without changing the source code
- Functions are **first-class citizens** in python, which means they can be:
  - ☐ passed as arguments to other functions
  - ☐ returned as values to other functions
  - ☐ defined within other functions
- The above properties validate decorators
- Example – "**@timeit**" decorator measures execution time of a code

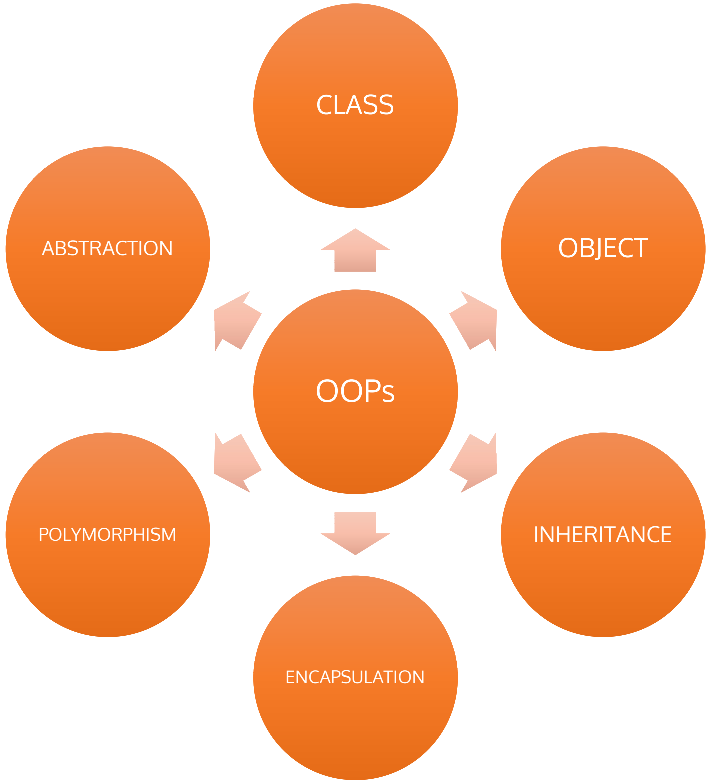
PYTHON PROGRAMMING: SPECIAL VARIABLES

- **Special Variables** can provide details on the context, behavior or state of a code in Python
    - Always defined using double underscores (e.g. “**__name__**”)
    - They are also known as dunder or magic methods
- **Examples:**
    - **__name__** contains the name of current module
        - __name__ shows '__main__' if the current script is running
        - If imported as a module into another script, __name__ shows the module's name
    - **__init__** is called when an object is created from a class and used to initialize the object's attributes
    - **__del__** is called for destroying an object and sending it for garbage collection
    - **__str__** is called by the str() function to generate a string that represents the object's state
    - **__call__** allows instances of a class to be called like functions
    - **__iter__** and **__next__** are used to define iterators for custom objects

anodiam

Education that enlightens!

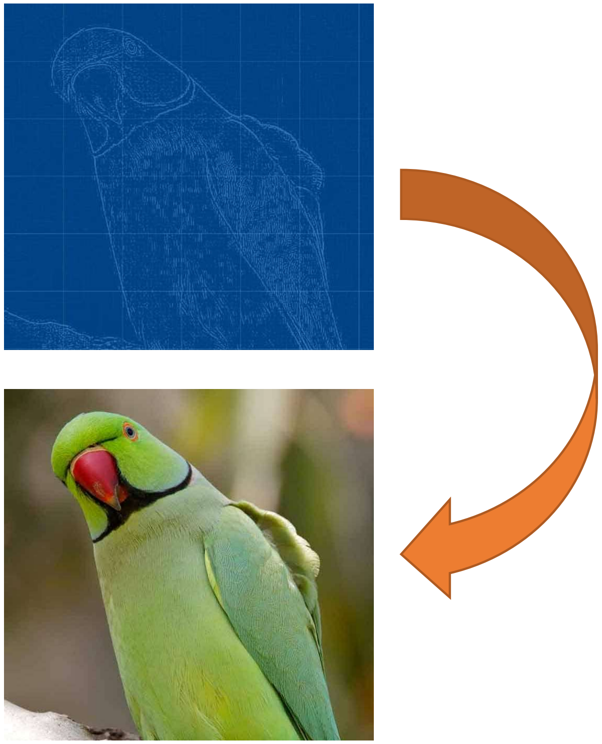
PYTHON PROGRAMMING: OBJECT ORIENTED PROGRAMMING


CLASS
ABSTRACTION
OBJECT
OOPs
POLYMORPHISM
INHERITANCE
ENCAPSULATION

- ❖ **OOPs** is a programming prototype that enables problem solving in a real world approach
    - ❑ Shows how versatile Python is, based on programming spectrum
    - ❑ Strings, Integers, Floats, Lists, Sets etc. are examples of built-in objects
    - ❑ To define an object, declaring a class is necessary
    - ❑ Classes contain **attributes** and **behaviors** of objects

Education that enlightens!

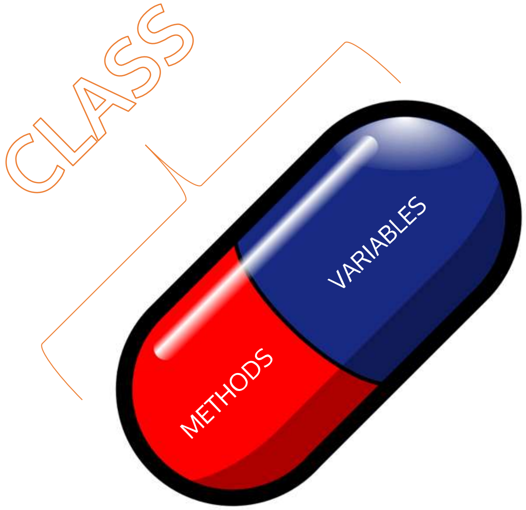
PYTHON PROGRAMMING: OBJECTS & CLASSES



- **Class** offers a blueprint for user-defined objects
  - Contains attributes(variables) and behaviors(methods) for the objects
  - Provides a way to organize code by grouping related data and behavior together
  - Keyword for creation is **class**
  - By convention class name should be typed in camel-case
- **Object** is an entity with **attributes** and **behaviors**
  - For example, an object **Parrot** has
    - **attributes** - age, color
    - **behavior** - flying, singing

  which belongs to the **class** "**Parrot**"
  - Objects are instances created from the class

PYTHON PROGRAMMING: CONSTRUCTORS & ENCAPSULATION



- **Constructors** initialize objects and allocate memory (in Heap Memory) to them
  - **__init__** method lets the class assign attributes for objects
  - **self** keyword (by convention) represents an instance (object) of the given class
  - Constructors are generally of two types
    - **Default** (Non-parameterized) does not accept any arguments except from **self**
    - **Parameterized** accepts arguments aside from **self**
- **Encapsulation** is a OOPs concept that ensures a conventional restriction of data
  - Wraps the variables and the methods together as a single unit
  - The idea is ensuring the variables can be accessed only through the methods of their current class
  - Constructors can be used for achieving encapsulation

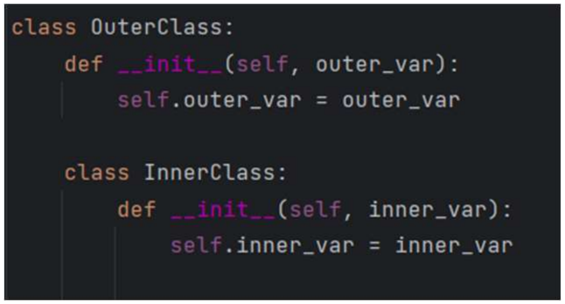
Python Programming: OOPs Variable Types

- **Instance Variables (Attributes)** are specific to an instance of a class
    - Stores unique data to each object created from the class
    - Object level
- **Class Variables (Static Variables)** are shared by all instances of a class
    - Defined within the class but outside of any instance methods
    - Same for all instances of the class (Class level)
    - Can be accessed using the class name or any instance of the class
- **Local Variables** are defined within methods or functions (Method level)
    - Only accessible within those
    - They have no connection to the class or its instances

Python Programming: OOPs Method Types

- ❖ **Instance Methods** are meant to access the instance attributes
    - ❑ Takes **self** as parameter
    - ❑ The most commonly used method
    - ❑ **Accessors** fetch (get) the value and **Mutators** modify (set) the value
- ❖ **Class Methods** are usually used to access class variables
    - ❑ Can be called using the class name itself
    - ❑ Takes **cls** as parameter
    - ❑ Declared by **@classmethod** decorator
- ❖ **Static Methods** are not associated with either of the class or instance variables
    - ❑ Takes no parameters
    - ❑ Declared by **@staticmethod** decorator
    - ❑ Can be accessed using the class name as well as class objects

PYTHON PROGRAMMING: INNER CLASS

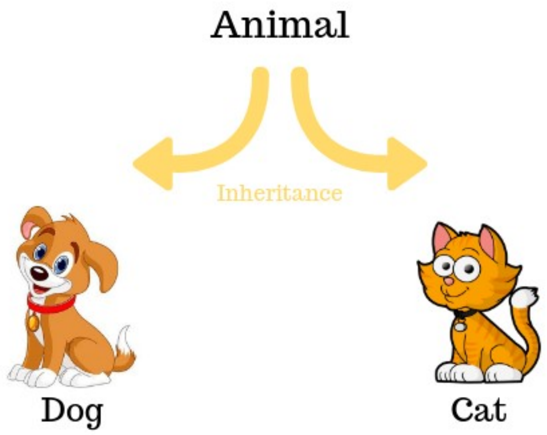


```
class OuterClass:
    def __init__(self, outer_var):
        self.outer_var = outer_var

    class InnerClass:
        def __init__(self, inner_var):
            self.inner_var = inner_var
```

- ❖ **Inner Classes** or nested classes are basically one defined inside the other
  - ❑ Useful for grouping two related classes
  - ❑ Better privacy of code
  - ❑ Inner classes are of two types, as follows
    - ➢ **Multiple:** The class contains one or more variables
    - ➢ **Multilevel:** The class contains an inner class which contains another inner class

PYTHON PROGRAMMING: INHERITANCE

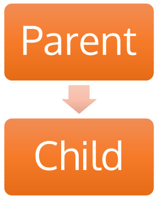


- ❖ **Inheritance** is a OOPs concept that allows a class to inherit attributes and behaviors from another class
  - ❑ Provides reusability of the code and increases its performance
  - ❑ **Parent Class (Super Class)** is the class whose properties are inherited by the new one
  - ❑ **Child Class (Sub Class)** derives its properties from its parent class aside from having its own

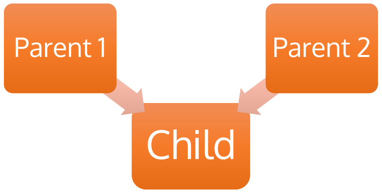
anodiam
Education that enlightens!

PYTHON PROGRAMMING: INHERITANCE TYPES

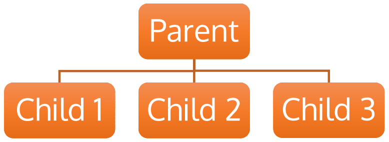
SINGLE LEVEL


Parent
Child

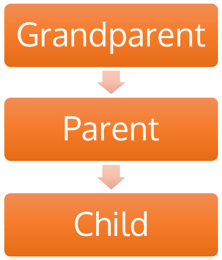
MULTIPLE


Parent 1
Parent 2
Child

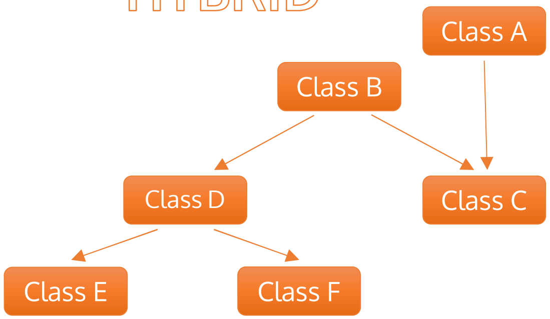
HIERARCHICAL


Parent
Child 1
Child 2
Child 3

MULTILEVEL


Grandparent
Parent
Child

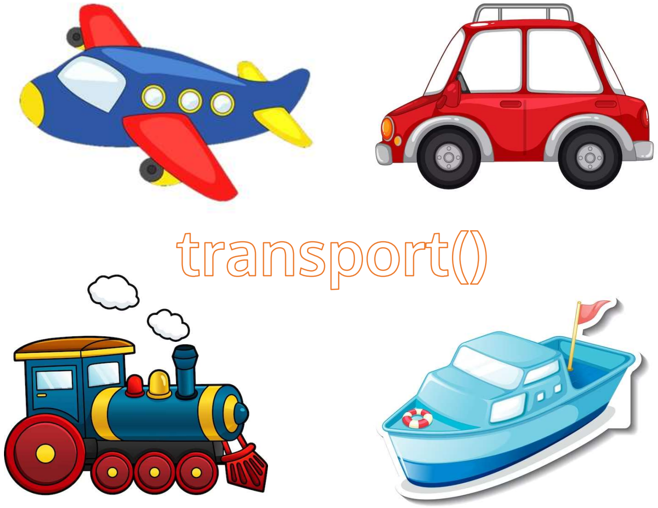
HYBRID


Class A
Class B
Class D
Class C
Class E
Class F

Python Programming: Method Resolution Order

- **MRO** is an order in which classes are searched when looking for a method or attribute in an inheritance hierarchy
    - Modern Python uses C3 Linearization algorithm over the old school DLR
    - **Example:** In Case of A(B,C) – MRO dictates that the code will first search A itself, then go to B and then C
    - **Left to Right**
- **Super()** is a built-in function which allows access to the parent's methods and attributes in the child class
    - Follows MRO

PYTHON PROGRAMMING: POLYMORPHISM

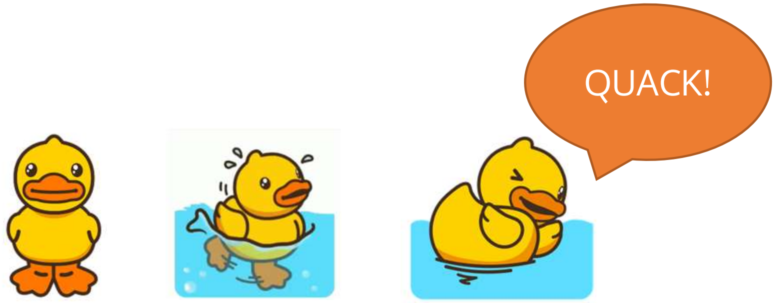


❖ **Polymorphism** means having multiple forms in general

- ❑ Provides code flexibility and reusability
- ❑ **len()** is an in-built polymorphic function
- ❑ Functions or methods with same name make different objects behave in similar pattern

PYTHON PROGRAMMING: DUCK TYPING

"If it looks like a duck, swims like a duck and quacks like a duck, then it probably is a duck"



- **Duck Typing** is an idea of assigning types to classes based on similar methods
    - Python is a dynamically typed language which dictates:
        - The type or class of an object is determined by its behavior
        - Declaring the type of an object explicitly is unnecessary

Anodiam

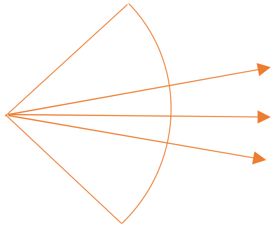
Education that enlightens!

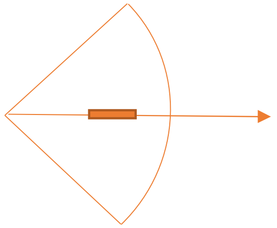
PYTHON PROGRAMMING: OPERATOR OVERLOADING

- ❖ **Operator Overloading** lets us define how operators behave for objects of custom classes
  - ❑ Operators in python actually calls certain special methods behind the scene
    - ➢ " **+** " calls __**add**__
    - ➢ " **-** " calls __**sub**__
    - ➢ " ***** " calls __**mul**__
  - ❑ Custom operations can be defined by modifying those special methods
    - ➢ For e.g. adding two objects by inducing new addition logic to __**add**__

Education that enlightens!

PYTHON PROGRAMMING: METHOD OVERLOADING & OVERRIDING



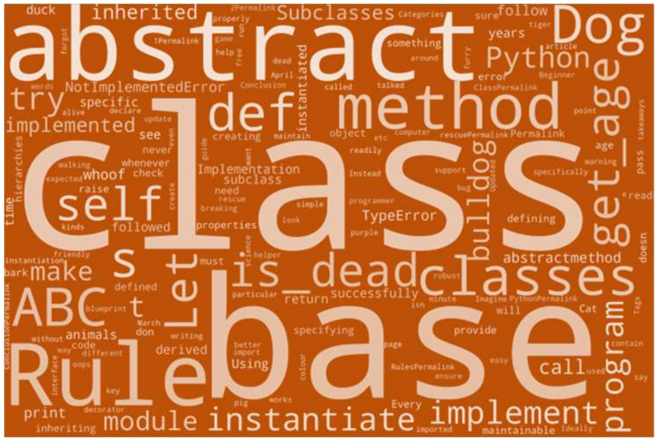


- ❖ **Method Overloading** defines multiple methods with the same name but different parameters
  - ❑ **Compile time** Polymorphism
  - ❑ Is implemented within the same class
  - ❑ Python does not support this by default which makes the latest definition valid for the overloaded method
- ❖ **Method Overriding** defines a new implementation for a method in a subclass that already exists in the parent class
  - ❑ **Run time** Polymorphism
  - ❑ Works within at least two classes (or more) related via inheritance
  - ❑ Methods must have same name and parameters

Anodiam

Education that enlightens!

PYTHON PROGRAMMING: ABSTRACT METHODS & CLASSES


abstract
class
base
method
def
Dog
Python
self
Rule
ABC
classes
is_dead
get_age
bulldog
program
implement
instantiate
module
Let
inherited
Subclasses
NotImplementedError
implemented
try
specific
Implementation
subclass
TypeError
abstractmethod
make
print
call
Using
derived
animals
whoof
followed
properties
return
successfully
specifying
provide
inheriting

- **Abstract Methods** are declared but not implemented
    - Defined using **@abstractmethod** decorator
    - Need to import **ABC** (Abstract Base Classes) module
    - Python does not support this by default
- **Abstract Classes** are classes that contain at least one abstract method
    - Used for building **API** (Application Program Interface)
    - Serves as a blueprint for other classes
    - Objects can not be created from them