Python

- Python is an interpreted, high-level and general-purpose, dynamically typed programming language
- It is also Object oriented, modular oriented and a scripting language.
- In Python, everything is considered as an Object.
- A python file has an extension of .py
- Python follows Indentation to separate code blocks instead of flower brackets({}).
- We can run a python file by the following command in cmd(Windows) or shell(mac/linux).

```
$ python <filename.py> or $ python3 <filename.py>
```

By default, python doesn't require any imports to run a python file.

Create and execute a program

1. Open up a terminal/cmd
2. Create the program: nano/cat > nameProgram.py
3. Write the program and save it
4. python nameProgram.py

Basic Datatypes

Data Type	Description
int	Integer values [0, 1, -2, 3]
float	Floating point values [0.1, 4.532, -5.092]
char	Characters [a, b, @, !, `]
str	Strings [abc, AbC, A@B, sd!, `asa]
bool	Boolean Values [True, False]
complex	Complex numbers [2+3j, 4-1j]

Keywords

- As of python3.8 there are 35 keywords

Keyword	Description	Category
True	Boolean value for not False or 1	Value Keyword
False	Boolean Value for not True or 0	Value Keyword
None	No Value	Value keyword
and	returns true if both (oprand) are true (other language &&)	Operator keyword
or	returns true of either operands is true (other language	
in	returns true if word is in iterator	Operator keyword
is	returns true if id of variables are same	Operator keyword
not	returns opposite Boolean value	Operator Keyword

Keyword	Description	Category
if	get into block if expression is true	conditional
elif	for more than 1 if checks	conditional
else	this block will be executed if condition is false	conditional
for	used for looping	iteration
while	used for looping	iteration
break	get out of loop	iteration
continue	skip for specific condition	iteration
def	make user defined function	structure
class	make user defined classes	structure
lambda	make anonymous function	structure
with	execute code within context manager's scope	structure
as	alias for something	structure
pass	used for making empty structures(declaration)	structure
return	get value(s) from function, get out of function	returning keyword
yield	yields values instead of returning (are called generators)	returning keyword
import	import libraries/modules/packages	import
from	import specific function/classes from modules/packages	import
try	this block will be tried to get executed	exception handling
except	is any exception/error has occured it'll be executed	exception handling
finally	It'll be executed no matter exception occurs or not	exception handling
raise	throws any specific error/exception	exception handling
assert	throws an AssertionError if condition is false	exception handling
async	used to define asynchronous functions/co-routines	asynchronous programming
await	used to specify a point when control is taken back	asynchronous programming
del	deletes/unsets any user defined data	variable handling
global	used to access variables defined outside of function	variable handling
nonlocal	modify variables from different scopes	variable handling

Operators

Operator	Description
( )	grouping parenthesis, function call, tuple declaration
[ ]	array indexing, also declaring lists etc.
!	relational not, complement, ! a yields true or false
~	bitwise not, ones complement, ~a
-	unary minus, - a
+	unary plus, + a
*	multiply, a * b
/	divide, a / b

Operator	Description
%	modulo, a % b
+	add, a + b
-	subtract, a - b
<<	shift left, left operand is shifted left by right operand bits (multiply by 2)
>>	shift right, left operand is shifted right by right operand bits (divide by 2)
<	less than, result is true or false, a %lt; b
<=	less than or equal, result is true or false, a <= b
>	greater than, result is true or false, a > b
>=	greater than or equal, result is true or false, a >= b
==	equal, result is true or false, a == b
!=	not equal, result is true or false, a != b
&	bitwise and, a & b
^	bitwise exclusive or XOR, a ^ b
|	bitwise or, a
&&, and	relational and, result is true or false, a < b && c >= d
||, or	relational or, result is true or false, a < b || c >= d
=	store or assignment
+=	add and store
-=	subtract and store
*=	multiply and store
/=	divide and store
%=	modulo and store
<<=	shift left and store
>>=	shift right and store
&=	bitwise and and store
^=	bitwise exclusive or and store
|=	bitwise or and store
,	separator as in (y=x,z=++x)

Basic Data Structures

List

- List is a collection which is ordered and changeable. Allows duplicate members.
- Lists are created using square brackets:

```
thislist = ["apple", "banana", "cherry"]
```

- List items are ordered, changeable, and allow duplicate values.
- List items are indexed, the first item has index `[0]`, the second item has index `[1]` etc.
- The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.
- To determine how many items a list has, use the `len()` function.
- A list can contain different data types:

```
list1 = ["abc", 34, True, 40, "male"]
```

- It is also possible to use the list() constructor when creating a new list

```
thislist = list(("apple", "banana", "cherry"))  # note the double round-brackets
```

- pop() function removes the last value in the given list by default.

```
    thislist = ["apple", "banana", "cherry"]

    print(thislist.pop())  # cherry
    print(thislist.pop(0))  #apple
```

Tuple

- Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
- A tuple is a collection which is ordered and unchangeable.
- Tuples are written with round brackets.

```
thistuple = ("apple", "banana", "cherry")
```

- Tuple items are ordered, unchangeable, and allow duplicate values.
- Tuple items are indexed, the first item has index `[0]`, the second item has index `[1]` etc.
- When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.
- Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.
- Since tuple are indexed, tuples can have items with the same value:
- Tuples allow duplicate values:

```
thistuple = ("apple", "banana", "cherry", "apple", "cherry")
```

- To determine how many items a tuple has, use the `len()` function:

```
thistuple = ("apple", "banana", "cherry")
print(len(thistuple))
```

- To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

```
thistuple = ("apple",)
print(type(thistuple))

# NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

- It is also possible to use the tuple() constructor to make a tuple.

```
thistuple = tuple(("apple", "banana", "cherry")) # note the double round-bracket
s
print(thistuple)
```

Set

- Set is a collection which is unordered and unindexed. No duplicate members.
- A set is a collection which is both unordered and unindexed.

```
thisset = {"apple", "banana", "cherry"}
```

- Set items are unordered, unchangeable, and do not allow duplicate values.
- Unordered means that the items in a set do not have a defined order.
- Set items can appear in a different order every time you use them, and cannot be referred to by index or key.
- Sets are unchangeable, meaning that we cannot change the items after the set has been created.
- Duplicate values will be ignored.
- To determine how many items a set has, use the `len()` method.

```
thisset = {"apple", "banana", "cherry"}

print(len(thisset))
```

- Set items can be of any data type:

```
set1 = {"apple", "banana", "cherry"}
set2 = {1, 5, 7, 9, 3}
set3 = {True, False, False}
set4 = {"abc", 34, True, 40, "male"}
```

- It is also possible to use the `set()` constructor to make a set.

```
thisset = set(("apple", "banana", "cherry")) # note the double round-brackets
```

- frozenset() is just an immutable version of Set. While elements of a set can be modified at any time, elements of the frozen set remain the same after creation.

```
set1 = {"apple", "banana", "cherry"}
frzset=frozenset(set1)
print(frzset)
```

Dictionary

- Dictionary is a collection which is unordered and changeable. No duplicate members.
- Dictionaries are used to store data values in key:value pairs.
- Dictionaries are written with curly brackets, and have keys and values:

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
```

- Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict["brand"])
```

- Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.
- Dictionaries cannot have two items with the same key.
- Duplicate values will overwrite existing values.

- To determine how many items a dictionary has, use the `len()` function.

```
print(len(thisdict))
```

- The values in dictionary items can be of any data type

```
thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
```

- pop() Function is used to remove a specific value from a dictionary. You can only use key bot the value. Unlike Lists you have to give a value to this function

```
 car = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = car.pop("model")

print(x)# Mustang
print(car)#{'brand': 'Ford', 'year': 1964}
```

Conditional branching

```
if condition:
    pass
elif condition2:
    pass
else:
    pass
```

Loops

Python has two primitive loop commands:

1. while loops
2. for loops

While loop

- With the `while` loop we can execute a set of statements as long as a condition is true.
- Example: Print i as long as i is less than 6

```
i = 1
while i < 6:
  print(i)
  i += 1
```

- The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.
- With the `break` statement we can stop the loop even if the while condition is true

- With the continue statement we can stop the current iteration, and continue with the next.
- With the else statement we can run a block of code once when the condition no longer is true.

For loop

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

- The for loop does not require an indexing variable to set beforehand.
- To loop through a set of code a specified number of times, we can use the range() function.
- The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.
- The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3).
- The else keyword in a for loop specifies a block of code to be executed when the loop is finished. A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
    print(x, y)
```

- for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

```
for x in [0, 1, 2]:
  pass
```

Function definition

```
def function_name():
    return
```

Function call

Python Versions

There are two main versions of Python called Python 2 and Python 3.

• Python 2 was launched in October 2000 and has been, and still is, very widely used.
• Python 3 was launched in December 2008 and is a major revision to the language that is not backward compatible.

The issue between the two versions can be highlighted by the simple print facility:

• In Python 2 this is written as

```
print 'Hello World'
```

• In Python 3 this is written as

```
print('Hello World')
```

Python 3 is the future of the Python language and it is this version that has introduced many of the new and improved language and library features (that have admittedly been back ported to Python 2 in many cases)

And this course is solely focussed on Python 3

Python Libraries

Many libraries available for Python.These libraries extend the functionality of the language and make it much easier to develop applications. These libraries cover

Some of the Python Standard Library modules

`collections` — Additional data structures beyond lists, tuples, dictionaries and sets.

`csv` — Processing comma-separated value files.

`datetime` , `time` — Date and time manipulations.

`decimal` — Fixed-point and floating-point arithmetic, including monetary calculations.

`doctest` — Simple unit testing via validation tests and expected results embedded in docstrings.

`json` — JavaScript Object Notation (JSON) processing for use with web services and NoSQL document databases.

`math` — Common math constants and operations.

`os` — Interacting with the operating system.

`queue` — First-in, first-out data structure.

`random` — Pseudorandom numbers.

`re` — Regular expressions for pattern matching.

`sqlite3` — SQLite relational database access.

`statistics` — Mathematical statistics functions like `mean` , `median` , `mode` and `variance` .

`string` — String processing.

`sys` — Command-line argument processing; standard input, standard output and standard error streams.

`timeit` — Performance analysis.

Popular Python libraries used in data science

Scientific Computing and Statistics

NumPy (Numerical Python)—Python does not have a built-in array data structure.

It uses lists, which are convenient but relatively slow. NumPy provides the high-performance `ndarray`

data structure to represent lists and matrices, and it also provides routines for processing such data structures.

SciPy (Scientific Python)—Built on NumPy, SciPy adds routines for scientific processing, such as integrals, differential equations, additional matrix processing and more. `scipy.org` controls SciPy and NumPy.

StatsModels—Provides support for estimations of statistical models, statistical tests and statistical data exploration.

Data Manipulation and Analysis

Pandas—An extremely popular library for data manipulations.

Pandas makes abundant use of NumPy's `ndarray` .

Its two key data structures are `Series` (one dimensional) and `DataFrames` (two dimensional).

Popular Python libraries used in data science

Visualization

Matplotlib—A highly customizable visualization and plotting library.

Supported plots include regular, scatter, bar, contour, pie, quiver, grid, polar axis, 3D and text.

Seaborn—A higher-level visualization library built on Matplotlib. Seaborn adds a nicer look-and-feel, additional visualizations and enables you to create visualizations with less code.

Machine Learning, Deep Learning and Reinforcement Learning

scikit-learn—Top machine-learning library. Machine learning is a subset of AI. Deep learning is a subset of machine learning that focuses on neural networks.

Keras—One of the easiest to use deep-learning libraries. Keras runs on top of TensorFlow (Google).

TensorFlow—From Google, this is the most widely used deep learning library. TensorFlow works with GPUs (graphics processing units) or Google's custom TPUs (Tensor processing units) for performance. TensorFlow is important in AI and big data analytics—where processing demands are huge. You'll use the version of Keras that's built into TensorFlow.

OpenAI Gym—A library and environment for developing, testing and comparing reinforcement-learning algorithms.

Natural Language Processing (NLP)

NLTK (Natural Language Toolkit)—Used for natural language processing (NLP) tasks.

TextBlob—An object-oriented NLP text-processing library built on the NLTK and pattern NLP libraries. TextBlob simplifies many NLP tasks.

Gensim—Similar to NLTK. Commonly used to build an index for a collection of documents, then determine how similar another document is to each of those in the index.

More Python Libraries

`Django/Flask` — web frameworks

`smtplib/imaplib` — EMail clients such as SMTP and IMAP4 email client.

`Python Excel Library` — Generation of Microsoft Excel Files.

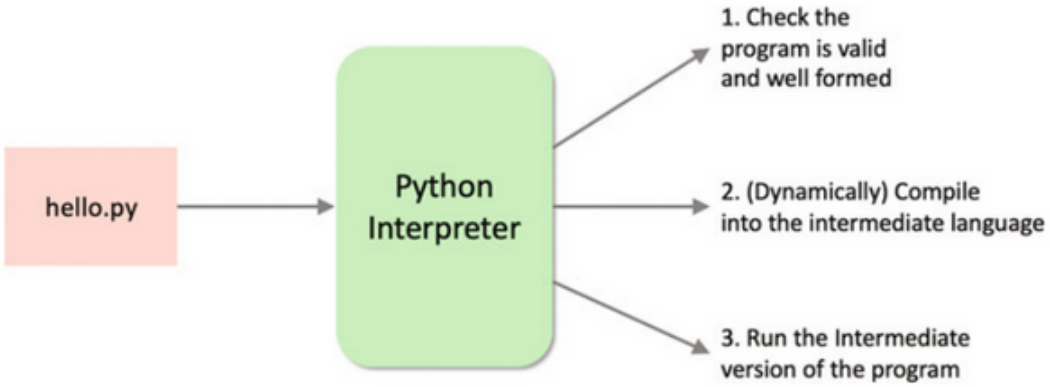
`Pygame` — Used for developing video games using computer graphics.

A very useful resource to look at, which introduces many of these libraries (also known as modules), is the '**Python 3 module of the Week**' web site which can be found at https://pymotw.com/3 (https://pymotw.com/3).

This lists many of the libraries/modules available and provides a short introduction to what they do and how

Python Execution Model

- Python is not a precompiled language in the way that some other languages you may have come across are (such as C++).
- Instead it is what is known as an **interpreted language** (although even this is not quite accurate)
- Python actually uses an intermediate model in that it actually converts the plain text English style Python program into an intermediate 'pseudo' machine code format and it is this intermediate format that is executed. This is illustrated below:



(https://ibb.co/7nFPyLN)

Steps:

1. First the program is checked to make sure that it is valid Python. That is a check is made that the program follows all the rules of the language and that each of the commands and operations etc. is understood by the Python environment.
2. It then translates the plain text, English like commands, into a more concise intermediate format that is easier to execute on a computer. Python can store this intermediate version in a file which is named after the original file but with a **'.pyc'** extension instead of a '.py' extension (the 'c' in the extension indicates it contains the compiled version of the code).
3. The compiled intermediate version is then executed by the interpreter.

A QUICK RECAP OF INTRO TO PYTHON PROGRAMMING

In [4]:
```
# variable 'x' is having value 10
x = 10

# multiple assignments
s , t = 11, 12  # here s value is 11 and t value is 12

# we use print() to display the value
print(x,s,t)
```

```
10 11 12
```

In [5]:
```
a = 12
b = 13

c = a
a = b
b = c

print(a,b)
```

```
13 12
```

In [6]:
```
a = 2
b = 4

a, b = b, a
print(a,b)
```

```
4 2
```

In [7]:
```
# print() has two arguments sep ( by default space)
# and end (by default is newline)
a,b,c,d = 11, 12, 13, 14
print(a,b,c, sep = "@", end = "#")
print(c)
print(d)
```

```
11@12@13#13
14
```

In [8]:
```
# mathematical operators
print(a+b*c-d**a/c+c//2+d%a)

# augmented assignment operators
a,b,c,d = 11, 12, 13, 14
a += 2  # a = 13
b -= 1  # b = 11
c *= 1  # c = 13
d /= a # d = d/a = 14/13 = 1.1
b %= 2  # b = b % 2 = 11%2 = 1
print(a,b,c,d)

# strings
h = "hello"
m = "to myself"
print(h + m + "!!!") # + is concatenation operator
```

```
-311505012875.0769
13 1 13 1.0769230769230769
helloto myself!!!
```

In [9]:
```
x = float("7")  # EXPLICIT CONVERSION
y = float("8")
print(x + y)
```

```
15.0
```

In [10]:
```
## adding two numbers
num1 = int(input("Enter a first number: "))
num2 = int(input("Enter a second number: "))
print("Adding of two numbers is: ", num1 + num2)
```

```
Enter a first number: 1
Enter a second number: 2
Adding of two numbers is:  3
```

In [11]:
```
a,b,c,d,e = 11, 12, 13, 14, 15
a += 2
b -= 1
c *= 1
d /= a # d = d/a = 14/13 = 1.1
e //= d
b %= 2  # b = b % 2 = 11%2 = 1
print(a,b,c,d,e)
```

```
13 1 13 1.0769230769230769 13.0
```

In [12]:
```
# to take input from the user
s = input("Enter a string (set of characters): ")
n = int(input("Enter a integer number: "))
f = float(input("Enter a decimal number: "))

# str() use to convert into string type to perform concatenation
print("String: " + s + " Integer Number: " + str(n) \
      + " Decimal Number: " + str(f))
```

```
Enter a string (set of characters): DPS Ruby
Enter a integer number: 786
Enter a decimal number: 06.09
String: DPS Ruby Integer Number: 786 Decimal Number: 6.09
```

In [13]:
```
# math module
# first import the library in your current session
import math

x = 9
z = math.sqrt(x) # calculate the square root of x
y = math.pow(x,2) # calculate the square of x
c = math.pi  # constant pi
d = math.fabs(y) # absolute value of y
e = math.log(2,3) # finds log 2 with base 3
f = math.log2(16) # value of log2 of 16
g = math.log10(10000) # value of log 10 of 10000
h = math.sin(c/6) # value of sin at pi/6
i = math.tan(c/6) # value of tan at pi/6
j = math.ceil(2.3) # least integer function
k = math.floor(2.3) # greatest integer function [x]

# print all the values
print(x,y,z,c,d,e,sep = " --> ")
print(f,g,h,i,j,k,sep = " --> ")
```

```
9 --> 81.0 --> 3.0 --> 3.141592653589793 --> 81.0 --> 0.6309297535714574
4.0 --> 4.0 --> 0.49999999999999994 --> 0.5773502691896257 --> 3 --> 2
```

In [14]:
```
import math

x, y = 9, math.pi
w, z, a = math.sqrt(x), math.pow(x,3), math.ceil(y)
print(w,z,a)
```

```
3.0 729.0 4
```

Python Character Set

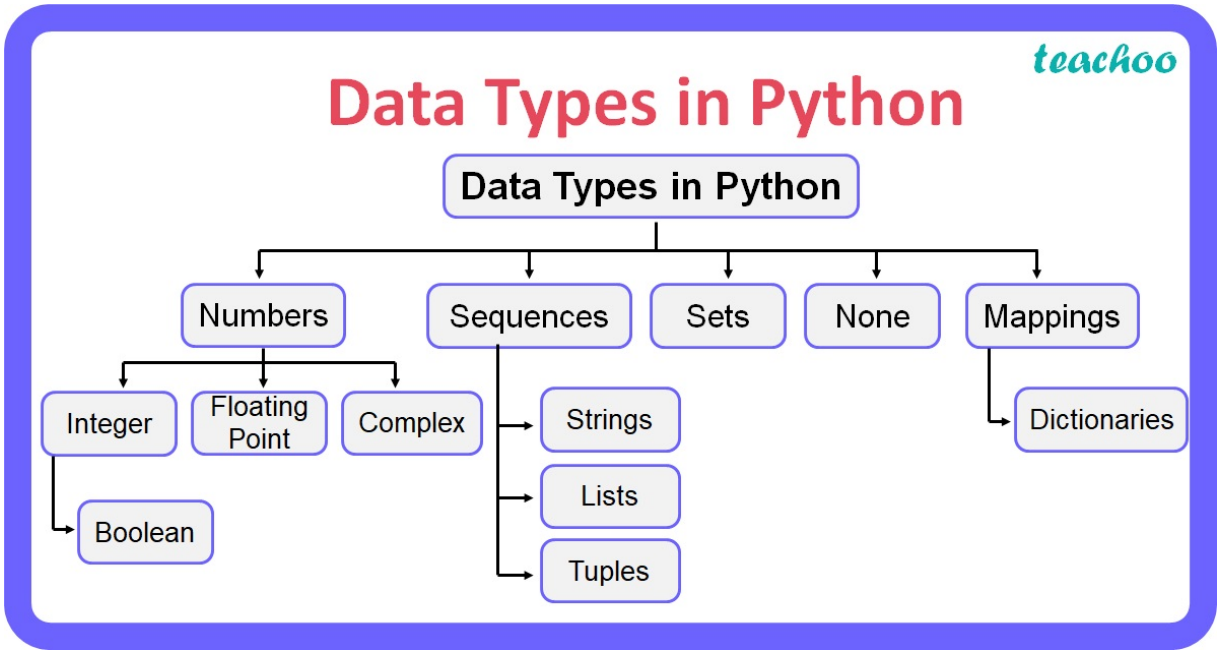
A character set is a set of valid characters acceptable by a programming language in scripting. In this case, we are talking about the Python programming language. So, the Python character set is a valid set of characters recognized by the Python language. These are the characters we can use during writing a script in Python. Python supports all ASCII / Unicode characters that include:

- **Alphabets**: All capital (A-Z) and small (a-z) alphabets.
- **Digits**: All digits 0-9.
- **Special Symbols**: Python supports all kind of special symbols like, ” ‘ I ; : ! ~ @ # $ % ^ ` & ( ) _ + – = { } [ ] \ .
- **White Spaces**: White spaces like tab space, blank space, newline, and carriage return.

- **Other**: All ASCII and UNICODE characters are supported by Python that constitutes the Python character set.

In [16]:
```
### Identity of the Variable/Object
### It refers to the object's address in the memory which does not change once it ha

x = 10
id(x)
```

Out[16]: 140722341467208



In [17]:
```python
# DataType Output: str
x = "Hello World"

# DataType Output: int
x = 50

# DataType Output: float
x = 60.5

# DataType Output: complex
x = 0 + 3j

# DataType Output: list
x = [1, 2, 3, 4]

# DataType Output: tuple
x = (1,2,3,4)

# DataType Output: range
x = range(10)

# DataType Output: dict
x = {"name": "Suraj", "age": 24, "IP marks": 50}

# DataType Output: set
x = {"gdgpss", "for", "gdgpss"}

# DataType Output: frozenset
x = frozenset({"gdgpss", "for", "gdgpss"})

# DataType Output: bool
x = True

# DataType Output: NoneType
x = None

```

In [18]:
```python
x = [1,2,3,4,4,5,5,5]
x
```

Out[18]: [1, 2, 3, 4, 4, 5, 5, 5]

In [19]:
```python
### COMPLEX NUMBER
```

In [20]:
```python
x = 2 + 5j
print(x.real, x.imag)
```

```
2.0 5.0
```

In [21]:
```python
x = 3 + 4j
y = 5j
print(x + y)
```

```
(3+9j)
```

In [22]:
```
y = 4 - 2j
print(y.real, y.imag)
```

```
4.0 -2.0
```

In [25]:
```
z = x*y
z
```

Out[25]:
```
(20+10j)
```

In [26]:
```
x = 5 - 2j
y = -6 + 1j
z = 3
print(x + y + z)
print(x.real + y.real + z.real)
print(x*z)
print(x*y)
```

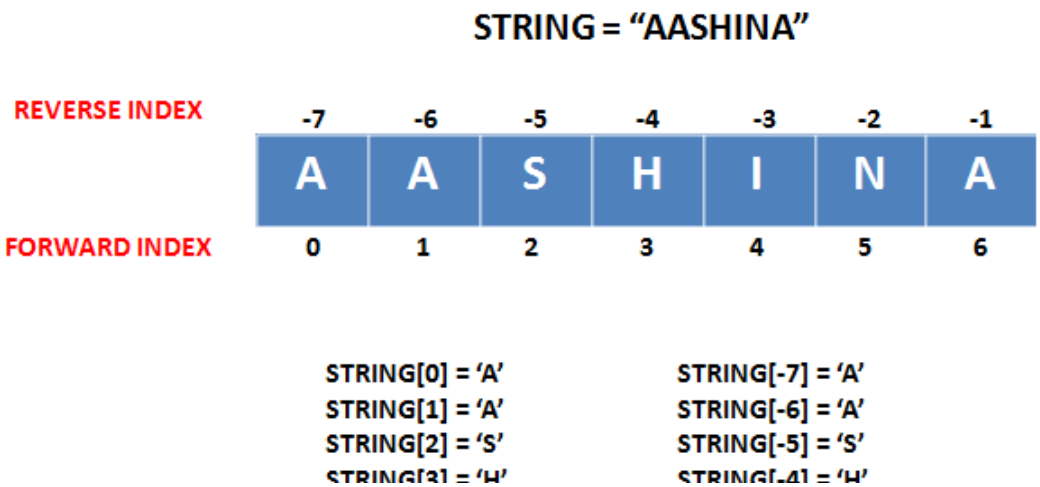
```
(2-1j)
2.0
(15-6j)
(-28+17j)
```

In [27]:
```
# Python code to demonstrate the working of
# complex(), real() and imag()

# importing "cmath" for complex number operations
import cmath

# Initializing real numbers
x = 5
y = 3

# converting x and y into complex number
z = complex(x,y);

# printing real and imaginary part of complex number
print ("The real part of complex number is : ",end="")
print (z.real)

print ("The imaginary part of complex number is : ",end="")
print (z.imag)

```

```
The real part of complex number is : 5.0
The imaginary part of complex number is : 3.0
```

STRING

- String is a sequence of characters and each character can be individually accessed using an Index.
- Strings are stored as individual characters in a contiguous location, with a 2-way index for each location.
- We can create them simply by enclosing the characters in quotes (eg:- ‘Hello’).
- Python treats single quotes the same as double-quotes.



In [37]:
```
1 s = "DPS Ruby Park Kolkata"
2 s
```

Out[37]: 'DPS Ruby Park Kolkata'

In [43]:
```
1 s = "DPSRPK"
2 print(s[0], s[-1], s[-2], s[4])
```

```
D K P P
```

In [44]:
```
1 s[2] = 'D'  #IMMUTABLE
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[44], line 1
----> 1 s[2] = 'D'  #IMMUTABLE

TypeError: 'str' object does not support item assignment
```

In [45]:
```
1 print(s[-5])
```

```
P
```

In [46]:
```
1 print(s[-6], s[4], s[4], s[-3])
```

```
D P P R
```

In [47]:
```
1 s = "SIMIGURI"
2 s[2] = "L"
3 s
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[47], line 2
      1 s = "SIMIGURI"
----> 2 s[2] = "L"
      3 s

TypeError: 'str' object does not support item assignment
```

You cannot change the individual letters of a string in place by assignment because strings are immutable and hence item assignment is not supported.

In [49]:
```
l = [10,2,3,4,5,6]
l[0] = 1
l

# LISTS AND DICTIONARY ARE MUTABLE DATA TYPE
```

Out[49]:
```
[1, 2, 3, 4, 5, 6]
```

Multiline String

- It is represented using **triple double quotes (""" """)** or even **triple single quotes (''' ''')**

In [50]:
```
a = """ This is a
multiline string
example """

print(a)
```

```
 This is a
multiline string
example
```

In [51]:
```
s = """ We are students
we study computer science
programming language is python """

print(s)
```

```
 We are students
we study computer science
programming language is python
```

PROGRAMMING QUESTIONS

COMPLEX NUMBER BASED

1. WAP to add, subtract, multiply, divide two complex number.
2. WAP to find the conjugate of a complex number.
3. WAP to find the magnitude of a complex number.
4. WAP to find the phase of a complex number.
5. WAP to find the roots of a quadratic equation with complex coefficients.
6. WAP to find the sum of a list of complex numbers.
7. WAP to sort a list of complex number by magnitude.
8. WAP to multply two complex number using the polar form.

SOLUTION

In [52]:
```
### SOLUTION

# 1
## Addition
c1 = complex(2, 3)
c2 = complex(4, 5)
c3 = c1 + c2  ## addition
c4 = c1 - c2  ## subtraction
c5 = c1*c2    ## multiply
c6 = c1/c2    ## divide
print("The sum of", c1, "and", c2, "is", c3)
print("The difference of", c1, "and", c2, "is", c4)
print("The multiplication of", c1, "and", c2, "is", c5)
print("The division of", c1, "and", c2, "is", c6)
```

```
The sum of (2+3j) and (4+5j) is (6+8j)
The difference of (2+3j) and (4+5j) is (-2-2j)
The multiplication of (2+3j) and (4+5j) is (-7+22j)
The division of (2+3j) and (4+5j) is (0.5609756097560976+0.0487804878048781j)
```

In [53]:
```
# 02  --> CONJUGATE
c = complex(2, 3)
conjugate = c.conjugate()
print("The conjugate of", c, "is", conjugate)
```

```
The conjugate of (2+3j) is (2-3j)
```

In [54]:
```
# 03  --> MAGNITUDE
import math
c = complex(2, 3)
magnitude = math.sqrt(c.real ** 2 + c.imag ** 2)
print("The magnitude of", c, "is", magnitude)
```

```
The magnitude of (2+3j) is 3.605551275463989
```

In [55]:
```
# 04  --> PHASE
import math
c = complex(2, 3)
phase = math.atan2(c.imag, c.real)
```

In [58]:

```python
# 05 --> QE SOLN WITH D < 0
import cmath

# Coefficients of the quadratic equation ax^2 + bx + c = 0
a = 1
b = -4
c = 5

# Calculate the discriminant
discriminant = b**2 - 4*a*c

# Check if the discriminant is negative
if discriminant < 0:
    # Calculate the complex roots
    root1 = (-b + cmath.sqrt(discriminant)) / (2*a)
    root2 = (-b - cmath.sqrt(discriminant)) / (2*a)

    print("Complex Roots:")
    print("Root 1:", root1)
    print("Root 2:", root2)
else:
    # Calculate the real roots
    root1 = (-b + cmath.sqrt(discriminant)) / (2*a)
    root2 = (-b - cmath.sqrt(discriminant)) / (2*a)

    print("Real Roots:")
    print("Root 1:", root1)
    print("Root 2:", root2)

```

```
Complex Roots:
Root 1: (2+1j)
Root 2: (2-1j)
```

In [60]:

```python
# 06  --> SUM OF LIST OF CN
nums = [complex(2, 3), complex(4, 5), complex(1, 1)]
sum = complex(0, 0)

for num in nums:
    sum += num
print("The sum of", nums, "is", sum)
```

```
The sum of [(2+3j), (4+5j), (1+1j)] is (7+9j)
```

In [61]:

```python
# 07  --> CN SORTED WITH MAGNITUDE VALUE
import math
nums = [complex(2, 3), complex(4, 5), complex(1, 1)]
sorted_nums = sorted(nums, key=lambda x: math.sqrt(x.real ** 2 + x.imag ** 2))
print("The sorted list of", nums, "by magnitude is", sorted_nums)
```

```
The sorted list of [(2+3j), (4+5j), (1+1j)] by magnitude is [(1+1j), (2+3j), (4+5j)]
```

In [62]:

```python
# 08  --> PLOAR FORM
import math
c1 = complex(2, 3)
c2 = complex(4, 5)

r1, phi1 = math.sqrt(c1.real ** 2 + c1.imag ** 2), math.atan2(c1.imag, c1.real)
r2, phi2 = math.sqrt(c2.real ** 2 + c2.imag ** 2), math.atan2(c2.imag, c2.real)

r3, phi3 = r1 * r2, phi1 + phi2

c3 = complex(r3 * math.cos(phi3), r3 * math.sin(phi3))

print("The product of", c1, "and", c2, "is", c3)
```

```
The product of (2+3j) and (4+5j) is (-6.999999999999996+22j)
```

TRIGONOMETRY BASED

1. WAP that calculates the sine, cosine, and tangent of an angle in degrees.
2. WAP to calculate the length of one side of a right triangle given the lengths of another side and an angle in the triangle.
3. WAP to calculate the inverse of sine, cosine, tangent of a number.
4. WAP to calculate the distance between two points on the surface of the earth.
5. WAP to calculate the angle between two vectors.
6. WAP to solve the direct kinematics problem for a simple robotic arm with two joints.
7. WAP to calculate the amplitude, period, and phase shift of a sine wave.
8. WAP to calculate the area of a regular polygon.

SOLUTION

In [74]:

```python
## 01
import math

# Input the angle in degrees from the user
angle_degrees = float(input("Enter the angle in degrees: "))

# Convert the angle from degrees to radians
angle_radians = math.radians(angle_degrees)

# Calculate sine, cosine, and tangent
sine = math.sin(angle_radians)
cosine = math.cos(angle_radians)
tangent = math.tan(angle_radians)

# Print the results
print(f"Sine of {angle_degrees} degrees: {sine:.4f}")
print(f"Cosine of {angle_degrees} degrees: {cosine:.4f}")
print(f"Tangent of {angle_degrees} degrees: {tangent:.4f}")
```

```
Enter the angle in degrees: 30
Sine of 30.0 degrees: 0.5000
Cosine of 30.0 degrees: 0.8660
Tangent of 30.0 degrees: 0.5774
```

In [65]:

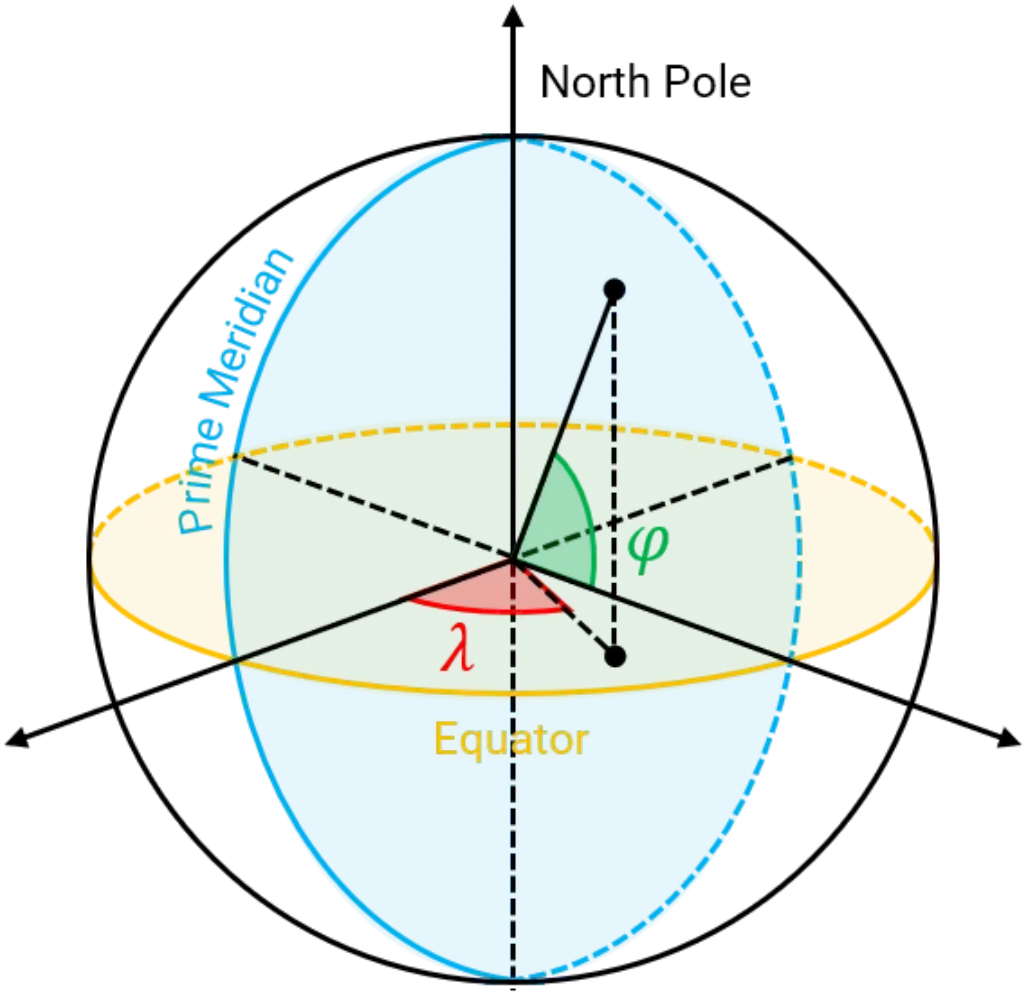
```python
## 02
import math

# Function to calculate the length of a side using trigonometry
def calculate_side_length(side, angle_degrees, hypotenuse):
    # Convert the angle from degrees to radians
    angle_radians = math.radians(angle_degrees)

    # Calculate the side length using trigonometry: side = hypotenuse * sin(angle)
    side_length = hypotenuse * math.sin(angle_radians)
    return side_length

# Input the length of the hypotenuse and an angle in degrees from the user
hypotenuse = float(input("Enter the length of the hypotenuse: "))
angle_degrees = float(input("Enter the angle in degrees: "))

# Calculate the length of the other side
other_side = calculate_side_length(None, angle_degrees, hypotenuse)

# Print the result
print(f"The length of the other side is {other_side:.2f}")
```

```
Enter the length of the hypotenuse: 5
Enter the angle in degrees: 30
The length of the other side is 2.50
```

In [66]:

```python
## 03
import math
num = 0.5

arc_sin = math.asin(num)
arc_cos = math.acos(num)
arc_tan = math.atan(num)

print("The arc sine of", num, "is", arc_sin, "radians")
print("The arc cosine of", num, "is", arc_cos, "radians")
print("The arc tangent of", num, "is", arc_tan, "radians")
```

```
The arc sine of 0.5 is 0.5235987755982989 radians
The arc cosine of 0.5 is 1.0471975511965979 radians
The arc tangent of 0.5 is 0.4636476090008061 radians
```

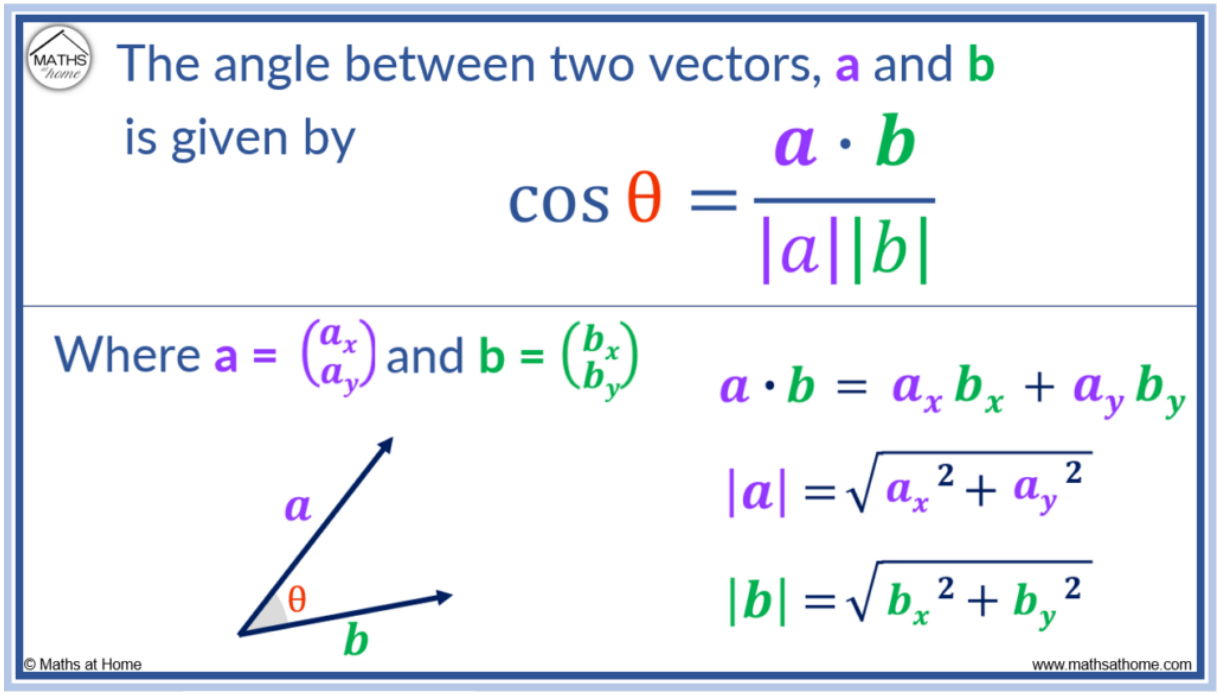


Link: [Haversine Formula (https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128#:~:text=All%20of%20these%20can%20be,longitude%20of%20the%20two%20points.)](https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128#:~:text=All%20of%20these%20can%20be,longitude%20of%20the%20two%20points.)

In [72]:

```python
## 04 <- Haversine Formula
import math

# define the radius of the Earth in kilometers
R = 6371

# define the coordinates of the two points in degrees
lat1, lon1 = 52.5200, 13.4050
lat2, lon2 = 48.8566, 2.3522

# convert the latitude and longitude from degrees to radians
lat1_radians = math.radians(lat1)
lon1_radians = math.radians(lon1)
lat2_radians = math.radians(lat2)
lon2_radians = math.radians(lon2)

# calculate the distance between the two points using the Haversine formula
delta_lat = lat2_radians - lat1_radians
delta_lon = lon2_radians - lon1_radians
a = math.sin(delta_lat / 2) ** 2 + math.cos(lat1_radians) * math.cos(lat2_radians) *
c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
distance = R * c

print("The distance between the points (", lat1, ",", lon1, ") and (", lat2, ",", lo
```

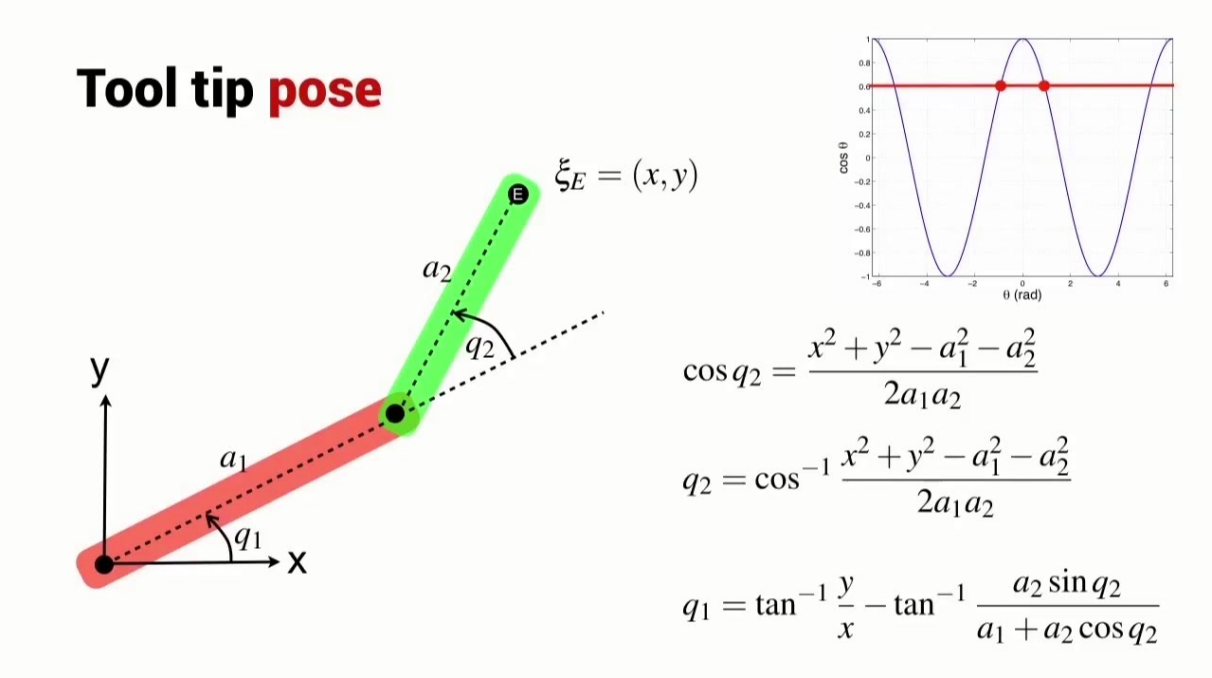
```
The distance between the points ( 52.52 , 13.405 ) and ( 48.8566 , 2.3522 ) is 877.4633
259175432 kilometers
```



In [69]:

```python
# 05
import math

# define the components of the two vectors
x1, y1 = 2, 3
x2, y2 = 5, -1

# calculate the dot product of the vectors
dot_product = x1 * x2 + y1 * y2

# calculate the magnitudes of the vectors
magnitude1 = math.sqrt(x1 ** 2 + y1 ** 2)
magnitude2 = math.sqrt(x2 ** 2 + y2 ** 2)

# calculate the angle between the vectors in degrees
angle = math.degrees(math.acos(dot_product / (magnitude1 * magnitude2)))

print("The angle between the vectors (", x1, ",", y1, ") and (", x2, ",", y2, ") is"
```

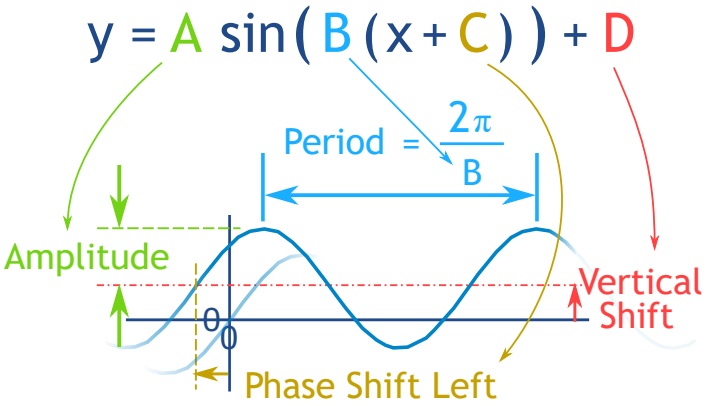
```
The angle between the vectors ( 2 , 3 ) and ( 5 , -1 ) is 67.61986494804043 degrees
```



In [70]:

```python
## 06
import math

# define the lengths of the two arm segments in meters
L1, L2 = 0.5, 0.5

# define the angles of the two joints in degrees
theta1, theta2 = 30, 45

# convert the angles from degrees to radians
theta1_radians = math.radians(theta1)
theta2_radians = math.radians(theta2)

# calculate the x and y coordinates of the end effector position
x = L1 * math.cos(theta1_radians) + L2 * math.cos(theta1_radians + theta2_radians)
y = L1 * math.sin(theta1_radians) + L2 * math.sin(theta1_radians + theta2_radians)

print("The end effector position for joint angles", theta1, "and", theta2, "is (", x
```

```
The end effector position for joint angles 30 and 45 is ( 0.5624222244434798 , 0.732962
9131445341 )
```



In [73]:

```python
## 07
import math

# define the parameters of the sine wave
A, B, C, D = 2, 1, 0, 1

# calculate the amplitude of the sine wave
amplitude = abs(A)

# calculate the period of the sine wave
period = 2 * math.pi / abs(B)

# calculate the phase shift of the sine wave
if B > 0:
    phase_shift = C - math.asin(D / A) / B
else:
    phase_shift = C + math.asin(D / A) / abs(B)

print("The amplitude of the sine wave is", amplitude)
print("The period of the sine wave is", period)
print("The phase shift of the sine wave is", phase_shift)
```

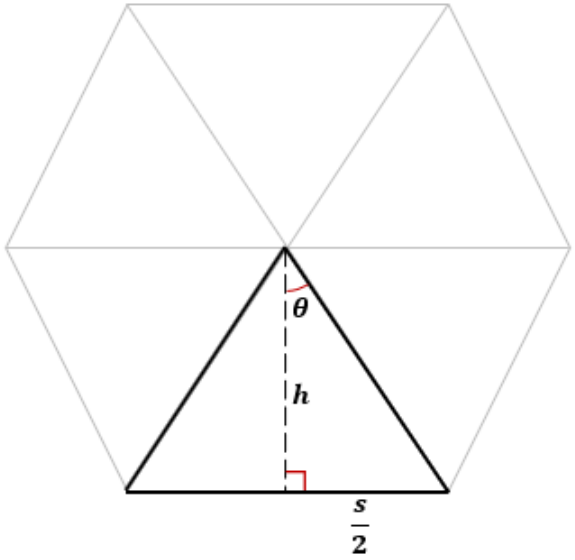
```
The amplitude of the sine wave is 2
The period of the sine wave is 6.283185307179586
The phase shift of the sine wave is -0.5235987755982989
```

Divide the regular polygon up into isosceles triangles as shown:

$$n = number\ of\ sides\ in\ regular\ polygon$$

$$\theta = \frac{180°}{n}$$

$$s = length\ of\ each\ side$$



$$\tan\theta = \frac{opposite}{adjacent} = \frac{\left(\frac{s}{2}\right)}{h}$$

$$h = \frac{s}{2\tan\theta}$$

$$Area\ of\ triangle = \left(\frac{1}{2}\right)bh = \left(\frac{s}{2}\right)\left(\frac{s}{2\tan\theta}\right)$$

General formula for *all* regular polygons:

$$Area\ of\ regualr\ polygon = \frac{ns^2}{4\tan\left(\frac{180°}{n}\right)}$$

In [95]:

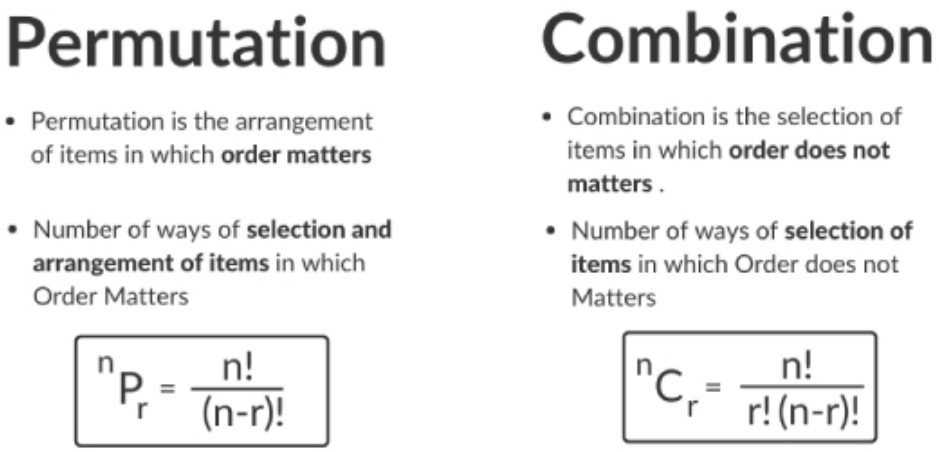
```
## 08
##
# Compute the area of a regular polygon.
#
from math import tan, pi
# Read input from the user
s = float(input("Enter the length of each side: "))
n = int(input("Enter the number of sides: "))

# Compute the area of the polygon
area = (n * s ** 2) / (4 * tan(pi / n))
# Display the result
print("The area of the polygon is", area)
```

```
Enter the length of each side: 3
Enter the number of sides: 6
The area of the polygon is 23.382685902179844
```

PERMUTATION, COMBINATION AND BINOMIAL THEOREM BASED

1. WAP to calculate the factorial of a number.
2. WAP to calculate the number of permutations of 'n' objects taken 'r' at a time.
3. WAP program to calculate the number of combinations of 'n' objects taken 'r' at a time.
4. WAP to calculate the binomial theorem.
5. WAP to print Pascal's triangle upto 'n' rows
6. WAP to calculate the (R+1)th term in an binomial expression.

SOLUTION



In [75]:

```
## 01
# Function to calculate the factorial of a number using recursion
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# Input a number from the user
num = int(input("Enter a number: "))

# Calculate and print the factorial
result = factorial(num)
print(f"The factorial of {num} is {result}")
```

```
Enter a number: 5
The factorial of 5 is 120
```

In [76]:

```python
# factorial of given number
def factorial(n):

    # single line to find factorial
    return 1 if (n==1 or n==0) else n * factorial(n - 1)

num = 5
print("Factorial of",num,"is",factorial(num))
```

```
Factorial of 5 is 120
```

In [77]:

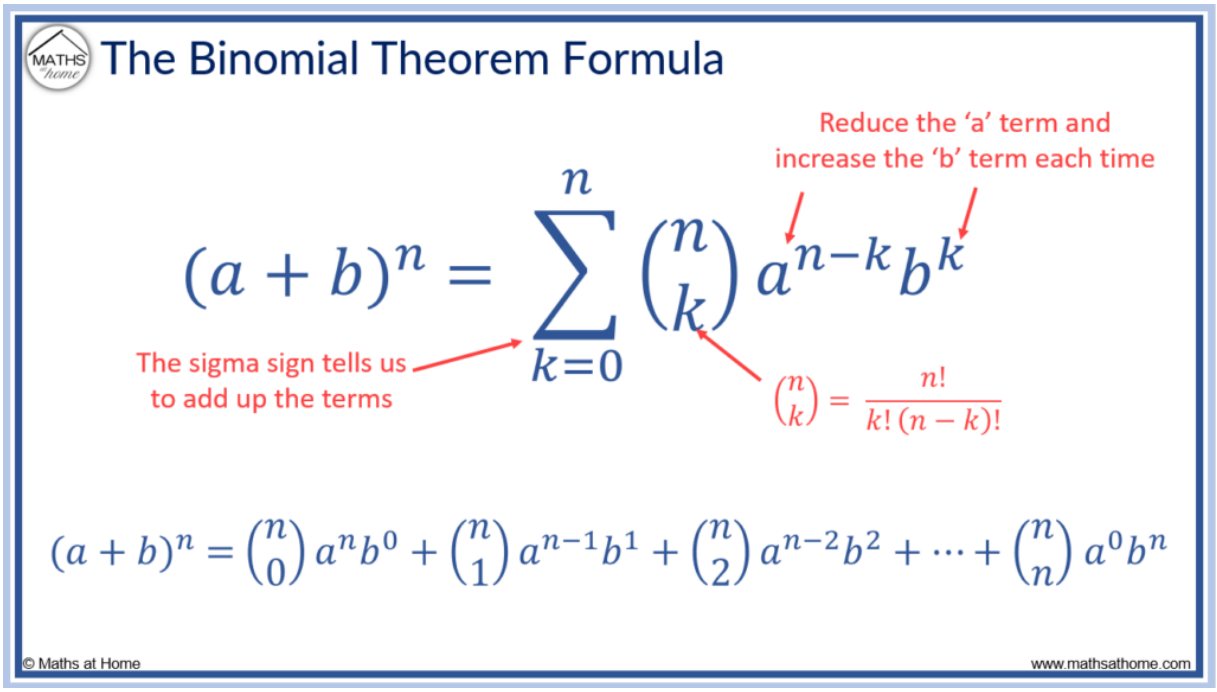
```python
## 02
import math

# Function to calculate permutations
def permutations(n, r):
    return math.factorial(n) // math.factorial(n - r)

# Input values of n and r from the user
n = int(input("Enter the value of n: "))
r = int(input("Enter the value of r: "))

# Calculate and print the number of permutations
result = permutations(n, r)
print(f"Number of permutations of {n} objects taken {r} at a time: {result}")
```

```
Enter the value of n: 5
Enter the value of r: 2
Number of permutations of 5 objects taken 2 at a time: 20
```

In [78]:

```python
## 03
import math

# Function to calculate combinations
def combinations(n, r):
    return math.factorial(n) // (math.factorial(r) * math.factorial(n - r))

# Input values of n and r from the user
n = int(input("Enter the value of n: "))
r = int(input("Enter the value of r: "))

# Calculate and print the number of combinations
result = combinations(n, r)
print(f"Number of combinations of {n} objects taken {r} at a time: {result}")
```

```
Enter the value of n: 3
Enter the value of r: 2
Number of combinations of 3 objects taken 2 at a time: 3
```



In [87]:

```python
## 04
import math

# function to calculate the binomial theorem
def binomial_theorem(a, b, n):
    expansion = ""
    for k in range(n+1):
        coefficient = math.comb(n, k)
        term = coefficient * (a**(n-k)) * (b**k)
        if k == 0:
            expansion += str(term)
        elif k == 1:
            expansion += " + " + str(term) + "x"
        else:
            expansion += " + " + str(term) + "x^" + str(k)
    return expansion

# get the values of a, b, and n from the user
a = int(input("Enter the value of a: "))
b = int(input("Enter the value of b: "))
n = int(input("Enter the value of n: "))

# calculate the binomial theorem
expansion = binomial_theorem(a, b, n)

print("The binomial theorem expansion is: ")
print(expansion)
```

```
Enter the value of a: 1
Enter the value of b: 2
Enter the value of n: 2
The binomial theorem expansion is:
1 + 4x + 4x^2
```

In [88]:

```python
## 05
## Pascals triangle

# function to print Pascal's triangle up to n rows
def print_pascals_triangle(n):
    # initialize the first row with 1
    row = [1]
    for i in range(n):
        # print the current row
        print(row)
        # calculate the next row
        next_row = [1]
        for j in range(len(row)-1):
            next_row.append(row[j] + row[j+1])
        next_row.append(1)
        row = next_row

# get the number of rows from the user
n = int(input("Enter the number of rows to print: "))

# print the Pascal's triangle
print_pascals_triangle(n)

```

```
Enter the number of rows to print: 4
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
```

In [94]:

```python
## 06
import math

# (a+b)^r
def binomial_term(a, b, r):
    coefficient = math.comb(r, 1) # binomial coefficient for the (r+1)-th term
    term = coefficient * (a**(r-1)) * (b**1) # formula for the (r+1)-th term
    return term

# get the values of a, b, and r from the user
a = int(input("Enter the value of a: "))
b = int(input("Enter the value of b: "))
r = int(input("Enter the value of r: "))

# calculate the (r+1)-th term of the binomial expansion
term = binomial_term(a, b, r)

print(f"The (r+1)-th term of the binomial expansion is: {term}")
```

```
Enter the value of a: 1
Enter the value of b: 1
Enter the value of r: 2
The (r+1)-th term of the binomial expansion is: 2
```

MORE INTERESTING PROGRAMMING QUESTIONS

Click Here (https://docs.google.com/presentation/d/e/2PACX-1vSHx8GuUZ-IWX3dqmpgGkLtczqMq-9ywZbEQyGzwA3MQtxO6f-TF-l2utijw7CqgTOfeInmXwm04Mj-/pub?start=false&loop=false&delayms=3000)