# Google Colaboratory

## Let's look at some common errors ……

*In the code cell, type following and click 'play' or other options to execute the code cell*

print (Hello world)

*You will 'sytax error'*

```
print(Hello world)

  File "<ipython-input-5-50b4ae29d403>", line 1
    print(Hello world)
                     ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

*You will see 'name error' if you type*

Print(a)

PRINT('Hello world')

```
print(a)

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-bca0e2660b9f> in <module>()
----> 1 print(a)

NameError: name 'a' is not defined
```

SEARCH STACK OVERFLOW

```
PRINT ('Hello world')

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-f749513eb754> in <module>()
----> 1 PRINT ('Hello world')

NameError: name 'PRINT' is not defined
```

SEARCH STACK OVERFLOW

# Google Colaboratory

## Let's look at some common errors ……

*In the code cell, type the following and click 'play' or other options to execute the code cell*

print ('Hello world')
print ('IDC101')

```
print('hello world')
print('IDC101')
```

```
hello world
IDC101
```

*Do the following:*

```
3+4
100+12
```

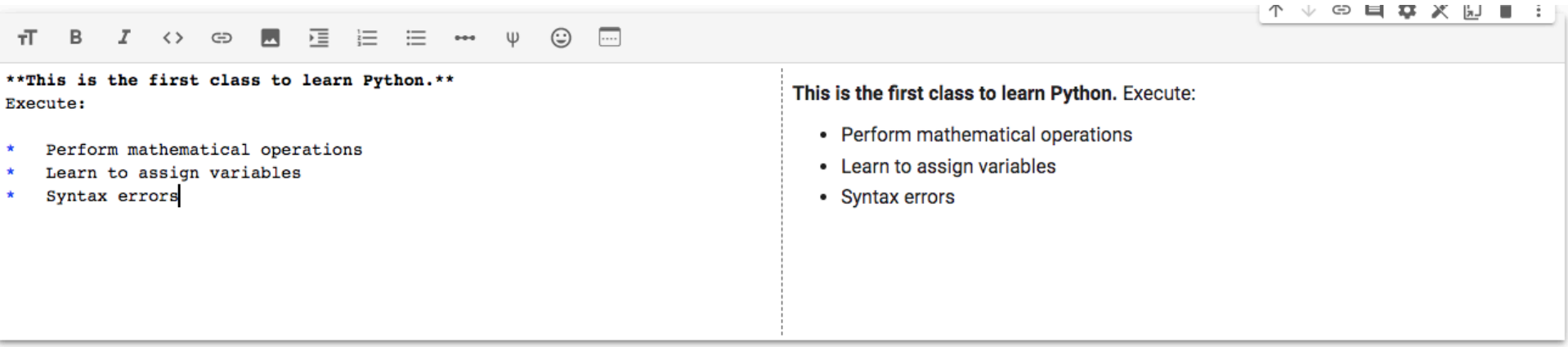
```
112
```

*Without assigning the result of mathematical operator to a variable, ONLY the result of last mathematical operation is shown.*

# Google Colaboratory

## Let's add Text

*Use "Text cell" to type whatever comment you want to add….*




*Material has been borrowed from Google Colab page*

# Mathematical operators

- *Use mathematical operators and observe the output.*
  - Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division (/)
  - Floor division (//)
  - Modulus (%)

- *Use relational operators*
  - Equals (==)
  - Greater than (>) or greater than or equal to (>=)
  - Less than (<) or less than or equal to (<=)

- *Assignment operator*
  - Assigns value from right side to left side

```
a = b
```

```
Value of b is assigned to a
```

## Syntax, statements and expression

1. Comments are essential part of coding used for:
   - Keeping a brief note on whole program or a particular statement.
   - Keeping key information as we tend to forget important purpose of a section/ statement in code.
   - Better understanding of a program or source code by anyone.

   It is always good practice to make properly comment/annotate your code. In Python, we can add comments in following ways:

   1. Use of (#) symbol to start writing a comment. By default it extends till the newline character (or return). Python Interpreter ignores the comment and does not execute it.

   2. Multiline string as comments is made by using text enclosed in a delimiter “”” …… “””. Importantly, there should be no white spaces between double quotes (“””). We can also use single quotes ‘’’….’’’

```
# This is a comment
a=19.91 # assignment operator
print(a) #print statement
```

```
""" This is a line
to demonstrate how to write comment
"""
a=19.91
print(a)
```

```
''' This is a line
to demonstrate how to write comment
'''
a=19.91
print(a)
```

## Syntax, statements and expression

Python program consists of **statements**. Statements performs some actions, executes something or changes the state. Instruction that a Python interpreter can execute is a statement. Statements can be:

- Assignment statements
- Expression statement
- Compound statements such as if-…-else

A statement can be extended into Multiple lines (or Multiline statement) using '\', ';', '[]', '{ }', '( )'. We will learn more about last 3 in later lectures.

```
x= 1+2+3+10+11+12+13+15+20
print(x)

x=1+2+3+10+11\
+12+13+15\
+20
print(x)
```

```
[ ] x=10
    y=20
    print(x+y)

    x=10;y=20;print(x+y)
```

```
x=10;y=20;z=20; print(x+
                      y
                      +x)
```

## Indentation

In Python, a block of code must be indented by same number of white spaces or a set of statements which need to be grouped together should have same indentation.

What is indentation? It is space referred at beginning of a statement.
What is block of code? The collections of statements, which will be executed as together when certain conditions are met.

A block of code must have indent having same number of white spaces (at least one). In other languages such as C/Perl/C++, it is usually done using curly braces {}.

```
a=True
b=2
if (a):
    print('This is first line of code block ')
    print('This is second line of code block ')
    while (b > 0):
        print('This is a line of code block WITHIN a code block ')
        print(b)
        b=b-1
```

Indentation

2nd level indentation (as it is code block within a code block)

## Indentation

```
for j in (0,1,2,3,8):
    print(j)
```

Indent

The below will give Syntax error as same block has statement (1) and statements (2) at *different levels of indentation. Fix the code below.*

```
for j in (0,1,2,3,8):
  1 print(j)
2 print(j+10)
```

## Variables

A variable is a named location used to store data in the memory. In some sense, you can think variable as something that holds data, which can be changed later or updated. Unlike many other languages, we do not have to define "type" of the variable. *Interestingly, a variable is created once we assign a value to it and type is inferred from it!!*

## Variables

Naming conventions for variables:

- It must begin with letters (A-Z,a-z) or underscore.
- It can consist of uppercase and lowercase letters (A-Z, a-z), digits (0-9), and the underscore character (_).
- Variable names are case sensitive, i.e. MYID and myID are not NOT same variable.
- Variables containing spaces is not correct.
- Never use special symbols like *!, @,#,$,&,*,*.
- Reserved words (keywords) cannot be used for naming the variable. Some of these are:

  *and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, if, while, for, try, with*

Correct ways of defining variable

```
Age=10
age=20
_age=5
_age_of_child=18
age_Of_Child=15
print(Age,age,_age,_age_of_child,age_Of_Child)
```

```
10 20 5 18 15
```

# Introduction to Python

Incorrect ways of defining variable. All of these will return SYNTAX error

```
7Age=10
print(7Age)
```

```
  File "<ipython-input-22-8c76df5a18a2>", line 1
    7Age=10
       ^
SyntaxError: invalid syntax
```

Starting variable with NUMBER is incorrect

```
for=10
```

```
  File "<ipython-input-23-c66d639d3bbe>", line 1
    for=10
       ^
SyntaxError: invalid syntax
```

Using reserved word for variable is incorrect

```
days ina week =7
```

```
  File "<ipython-input-24-e2627d47dd24>", line 1
    days ina week =7
            ^
```

Using spaces between variable is incorrect

# Introduction to Python

1. Variable can initialized and assigned using operator ('=').
2. Variables can re-assigned.
3. Accessing variable, which is not defined before will <u>return an error.</u>
4. Multiple variables can be initialized in one SINGLE assignment of values. The number of variables and their assigned numbers/strings must match in LHS and RHS.

(1)

```
i=34
print(i)
j=15
print(j)
```

```
34
15
```

(2)

```
i=34
print(i)
j=15
print(j)
j=20
print(j)
```

```
34
15
20
```

(4)

```
[35] a,b,c=10,20,30
     print(a,b,c)
```

```
10 20 30
```

(4)

```
a=b=c=50
print(a,b,c)
```

```
50 50 50
```

(3)

```
print(idc)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-32-1f650164205e> in <module>()
----> 1 print(idc)
```

Code with fixed error

```
idc='101'
print(idc)
```

```
101
```

## Basic data types

1. Numeric: is data which has numeric values. It can be Integers (without decimal part), floating point numbers, complex numbers.

   Integers: represented by 'int'. Contains negative/positive whole numbers (without fractional/decimal part)

   Floats: represented by 'float'. Real numbers with floating point representation.

   Complex number: it is specified by a real part and imaginary part (1+2j)

2. Strings: The type is represented by 'str'. The string is a sequence of SINGLE character(s). So a string is defined as one or many characters put between a single (single)/double quotes/ triple (single) quote.

3. Boolean: It has only two values 'True' and 'False'

## Basic data types (Numeric)

```
a=145
print(a)

b=0b1100
print(b)

c=0o10
print(c)

d=0xAB
print(d)
```

Binary number

Octal number

Hexadecimal number

```
145
12
8
171
```

```
a=1.234
print(a)

b=45.0
print(b)
```

```
1.234
45.0
```

```
a=1j
b=1j
c=a*b
print(c)
```

```
(-1+0j)
```

Binary number is prefixed with 0b/0B
Octal number is prefixed with 0o/0O
Hexadecimal number prefixed with 0x/0X

## Basic data types (String and Boolean)

```
a='I am enrolled in IDC101'
b=" I am learning Python"
c=''' Python is Fun'''

print(a)
print(b)
print(c)
```

```
I am enrolled in IDC101
 I am learning Python
 Python is Fun
```

```
a=True
b=False

print(a)
print(b)
```

```
True
False
```

## Types and conversion

In Python, all data has an associated data "Type". The type of data can be found by using a built-in-function *type().* You can also convert between types using:

*int()- converts compatible type to integer*
*float()- converts compatible type into float*
*str() – converts compatible type into string*

## Types

```python
a=10
print(str(a),"is of", type(a))

b=8.5
print(str(a),"is of",type(b))

c='IDC101'
print(c,"is of",type(c))
```

```
10 is of <class 'int'>
10 is of <class 'float'>
IDC101 is of <class 'str'>
```

```python
a=10
a_fl=float(a) # Type conversion to float
print('The type of',str(a),"is converted to float and new type is",  type(a_fl))
a_st=str(a) # Type conversion to String
print('The type of',str(a),"is converted to string and new type is ", type(a_st),'\n')

b=8.5
b_in=int(b) # Type conversion to int
print('The type of',str(b),"is converted to integer and new type is",  type(b_in))
b_st=str(b) # Type conversion to String
print('The type of',str(b),"is converted to string and new type is", type(b_st),'\n')

d=False
d_in=int(d) # converts bool to integer
d_fl=float(d) # converts bool to float
print('New value and types:', d_in,type(d_in))
print('New value and types:', d_fl,type(d_fl),'\n')
```

*Returns*

```
The type of 10 is converted to float and new type is <class 'float'>
The type of 10 is converted to string and new type is  <class 'str'>

The type of 8.5 is converted to integer and new type is <class 'int'>
The type of 8.5 is converted to string and new type is <class 'str'>

New value and types: 0 <class 'int'>
New value and types: 0.0 <class 'float'>
```

## Types

```
c='IDC101'
c_in=int(c) # incorrect compatible type for conversion. It will return error.
c_fl=int(c) # incorrect compatible type for conversion. It will return error
print(c,"is of",type(c))
```

*Returns*

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-105-8d2355985542> in <module>()
     19
     20 c='IDC101'
---> 21 c_in=int(c) # incorrect compatible type for conversion. It will return error.
     22 c_fl=int(c) # incorrect compatible type for conversion. It will return error
     23 print(c,"is of",type(c))
```

## Strings

Strings:

String is a sequence of SINGLE character(s). It is essentially array of bytes, which represent unicode character. Strings are **immutable** sequence of characters.

Immutable: are those those objects whose content cannot be changed or altered after creation (such as string, int, float, tuple)

Mutable: Objects can change their content such as lists, dict, set

Individual character in the String can be accessed by method Indexing. It starts from 0 to the (length-1) position. Negative index allows accessing string from last character starting with -1.

**Indexes are integers and should not exceed length of string.**

```
strTest='IDC 101'
print(strTest)
```

```
IDC 101
```

Index->

| I | D | C | | 1 | 0 | 1 |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| -7 | -6 | -5 | -4 | -3 | -2 | -1 |

<-Index

## id()

In python, id() is in-built function, which returns identity of an object. This is unique and remains constant for the object during the program. It can used to identify whether two objects are same or not.

```
a=5
print(id(a))
a=a+1
print(id(a))
```

```
94762711956096
94762711956128
```

```
a='IDC101'
b=a
print(id(a),id(b))
```

```
140426669059632 140426669059632
```

```
a='IDC101'
b=a
id(a) == id(b)
```

```
True
```

```
[28] a=257
     b=257
     id(a)==id(b)
```

```
False
```

## Strings

Example of indexing

```
strTest='IDC 123'
print(strTest)
```

```
[6] print(strTest[1]) ## Accessing second character in string strTest,
    print(strTest[-2]) ## Accessing second last character in the StrTest

    D
    2
```

If you try to access an index is more than length, you will find 'IndexError'.

```
print(strTest[8])

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-7-c70aafa45b40> in <module>()
----> 1 print(strTest[8])

IndexError: string index out of range

SEARCH STACK OVERFLOW
```

## Strings

If you try and change a character in the string, you will get 'TypeError'

```
strTest[2]='p'

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-8ec2aca88d9a> in <module>()
----> 1 strTest[2]='p'

TypeError: 'str' object does not support item assignment

SEARCH STACK OVERFLOW
```

If you want to extract a section of a string, you SLICE (take a part of) using the following option: STRING[x:y], this will return character from $X^{th}$ position till ($Y^{th}$-1) position.

```
a=strTest[1:3] ## returns character from 1st till 2nd position, which is DC"
print(a)
```

```
DC
```

## Strings

- Concatenating a string. You can use '+' operator. There are other ways to combine strings as well ……..

```
a='IDC 101'
b=' Introduction to computers'
print(a+b)
```

```
IDC 101 Introduction to computers
```

-Taking input from users using function *'input'*

**Content you want from user**

```
x=input('What is your name: ')
print(type(x))
print(x)
```

```
What is your name:
```

```
x=input('What is your name: ')
print(type(x))
print(x)
```

```
What is your name: Computer
<class 'str'>
Computer
```

## Strings (Slicing)

If you want to extract a section of a string, you SLICE (take a part of) using the slice extended indexing syntax. This indexing syntax works with other collection data types as well such as lists/tuple. We will learn about these in later classes.

Length of string is given by: len(string)

```
Slicing syntax is:
string[startIndex:stopIndex:step]

If startIndex is omitted (not specified) it defaults to start of string
If stopIndex is omitted (not specified) it defaults to rest of string
If step is not specified, it defaults to 1.
```

Index->

| I | D | C | | 1 | 0 | 1 |
|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| -7 | -6 | -5 | -4 | -3 | -2 | -1 |

<-Index

- ```
  a='IDC 101'
  print(a[:]) # All are set to default values
  ```
  ```

## Strings (Slicing)

- a=‘IDC 101’
  print(a[::]) # All are set to default values

string[:stopIndex] – returns string slice from start position till (stopIndex-1) position
String[startIndex:] - returns string slice from startIndex till last index position.
string[startIndex:stopIndex] – returns string slice from startIndex till (stopIndex-1) position

- print(a[:2])
- print(a[3:])
- print(a[0:7])

## Strings (Slicing)

- a=‘IDC 101’
- print(a[0:len(a):2])
- print(a[3:])
- print(a[0:7])

| | | | | | | |
|---|---|---|---|---|---|---|
| I | D | C | | 1 | 0 | 1 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| -7 | -6 | -5 | -4 | -3 | -2 | -1 |

- print(a[:-5]) ## the step size is 1 and startIndex defaults to start position in -7!!
- print(a[-5:]) ## stopIndex defaults to end position!!
- print(a[-5:-1:1]) ## Step size is 1.
- print(a[-1:-6]) ## Step size is 1.

## Strings (Slicing)

- `print(a[-1:-6:-1])`
- `print(a[-1::-1])`

## String methods

- string.capitalize() # return string with make first letter capital
- string.lower() # return lower case
- string.upper() # return upper case
- string.title() # return title case

Finding occurrence of a character/substring in a string can be searched by using '*in*'

## String methods example

```python
a='IDC 101 Introduction to computers'
print(a.capitalize()) # Make first letter capital rest all in small case
print(a.lower()) ## Make all letters in lower case
print(a.upper()) ## Make all letters in upper case
print(a.title()) ## Make in title case (all words starts with capital letter)
```

```
Idc 101 introduction to computers
idc 101 introduction to computers
IDC 101 INTRODUCTION TO COMPUTERS
Idc 101 Introduction To Computers
```

```python
'I' in a
```

```
True
```

```python
'computer' in a
```

```
True
```

```python
'coma' in a
```

```
False
```