In [2]:

```
from IPython.display import Image
Image(filename='logo.png', height=340, width=900)
```

Out[2]:

01 - PYTHON BASICS

WHAT IS PYTHON

Python is a popular programming language. It was created by Guido van Rossum and released in 1991

HISTORY OF PYTHON

Python is a widely used high-level programming language for general purpose programming, created by **`Guido van Rossum`** and first released in **`1991`**

Guido van Rossum was big fan of **`Monty Python's Flying Circus`** and hence had given the name of this programming language as Python.

Python uses a syntax that allows programmers to express concepts in fewer lines of code

Guido van Rossum released the first version (V.0.9.0) of the python code in February 1991. This release included already exception handling, functions, and the core data types of list, dictionary, strings and others. It was also object oriented and had a module system.

Python version 1.0 was released in January 1994. The major new features included in this release were the functional programming tools lambda, map, filter and reduce, which Guido Van Rossum never liked.

Six and a half years later in October 2000, Python 2.0 was introduced. This release included list comprehensions, a full garbage collector and it was supporting unicode.

Python flourished for another 8 years in the versions 2.x before the next major release as Python 3.0 (also known as "Python 3000" and "Py3K") was released.

WHERE IS PYTHON USED

- **To create web applications**
- **Used alongside software to create workflows**
- **Connect to database systems. It can also read and modify files**
- **To handle big data and perform complex mathematics**
- **Rapid prototyping, or for production-ready software development**

FEATURES OF PYTHON

- Easy-to-learn
- Easy-to-read
- Easy-to-maintain
- Interactive Mode
- Extendable
- Databases
- Scalable

IDE FOR PYTHON – INTEGRATED DEVELOPMENT ENVIRONMENT

1. Introduction of Anaconda as a distribution platform
2. Tools within Anaconda
3. Installation of Anaconda

BENEFITS OF ANACONDA

- Conda is an open source package management system and environment management system that runs on Windows, macOS and Linux
- Conda quickly installs, runs and updates packages and their dependencies.
- Conda easily creates, saves, loads and switches between environments on your local computer.
- It was created for Python programs, but it can package and distribute software for any language.
- Conda as a package manager helps you find and install packages
- The conda package and environment manager is included in all versions of Anaconda
- If we install Anaconda – Spyder, Jupyter, and even Python latest version would also be installed simultaneously.
- To install Anaconda -> Click on to the following links **`https://www.anaconda.com/download/`**
- Anaconda package which provides code completion and linking for Python and takes care of the basics, there are tons of themes, lightning fast user interface, easy configuration, tons of packages for more power features

CREATING VARIABLES

A variable is created the moment you first assign a value to it

INPUT VARIABLES

1. A variable name must start with a **`letter`** or the **`underscore`** character
2. A variable name **CANNOT** start with a **`number`**
3. A variable name can only contain **`alpha-numeric characters and underscores`**
4. A variable names are **`case-sensitive`**

OUTPUT VARIABLES

Print as an output variable

In [3]:

```
x= 5       #(which is a numeric data-type)
y="John"   #(which is a string data-type)
z=True     #(which is a boolean data-type)
print (x)
print (y)
print (z)
```

```
5
John
True
```

In [4]:

```
x=10
```

In [5]:

```
print(x)
```

```
10
```

In [6]:

```
x = 5
print (x)
```

```
5
```

In [7]:

```
X = 7
print(X)
print(x)
```

```
7
5
```

DATA TYPES IN PYTHON

There are the following basic data-types in python. To verify the type of any object in Python, use the **`type()`** function:

`print(type(variablename))`

1. **NUMERIC DATA TYPES:**

   `Integer`: Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length. **Example: `1`, `22222222222`, `-3876867432`**

   `Float`: Float, or "floating point number" is a number, positive or negative, containing one or more decimals. **Example: `1.0`, `1.157`, `-1.455`**

1. **STRINGS DATA TYPES:** String literals in python are surrounded by either single quotation marks, or double quotation marks. **Example: `a="Python is a language"`**

1. **BOOLEAN DATA TYPES:** There can only be two values to such data-type: **`TRUE` or `FALSE`**

In [8]:

```
print(type(x))
print(type(y))
print(type(z))
```

```
<class 'int'>
<class 'str'>
<class 'bool'>
```

In [9]:

```
type(x)
type(y)
type(z)
```

Out[9]:

```
bool
```

OPERATORS IN PYTHON

Operators are used to perform operations on variables and values. In python, there are following basic operators:

1. **`Arithmetic Operators`**
2. **`Assignment Operators`**
3. **`Comparison Operators`**
4. **`Logical Operators`**
5. **`Identity Operators`**
6. **`Membership Operators`**

ARITHMETIC OPERATORS

These operators are used to perform **`basic/common mathematical operations`**

Operator	Name	Action
+	Addition	`x+y`
-	Subtraction	`x-y`
*	Multiplication	`x*y`
/	Division	`x/y`
%	Modulus	`x%y`
**	Exponentiation	`x**y`
//	Floor Division	`x//y`

In [10]:

```
2+3
```

Out[10]:

```
5
```

In [11]:

```
3-5
```

Out[11]:

```
-2
```

In [12]:

```
3*8
```

Out[12]:

```
24
```

In [13]:

```
24/2
```

Out[13]:

```
12.0
```

In [14]:

```
2**6
```

Out[14]:

64

In [15]:

```
5//4
```

Out[15]:

1

In [16]:

```
6//4
```

Out[16]:

1

In [17]:

```
8//4
```

Out[17]:

2

In [18]:

```
6%4
```

Out[18]:

2

In [19]:

```
7%4
```

Out[19]:

3

Math expressions in Python follow the normal order of operations so * and / are executed before + and -, and ** is executed before multiplication and division.

In [20]:

```
# These operations are executed in reverse order of appearance.
2+3*5**2
```

Out[20]:

77

You can use parentheses in your math expressions to ensure that operations are carried out on the correct order. Operations within parentheses are carried out before operations that are external to the parentheses, just like you'd expect.

In [21]:

```
# Now the addition comes first and the exponentiation comes last.
```

```
((2+3) * 5 )**2
```

Out[21]:

625

ASSIGNMENT OPERATORS

These operators are used to assign values to variables

Operator	Example	Meaning
=	x=9	`x = 9`
+=	x+=9	`x += 9`
-=	x-=9	`x -= 9`
* =	x*=9	`x *= 9`
/=	x/=9	`x /= 9`
%=	x%=9	`x %= 9`
** =	x**=9	`x**=9`
//=	x//=9	`x//=9`

In [22]:

```
a = 21

a += 9
print ("Line 1 - Value of a is ", a)

a-=9
print ("Line 2 - Value of a is ", a)

a*=9
print ("Line 3 - Value of a is ", a)

a/=9
print ("Line 4 - Value of a is ", a)

a%=9
print ("Line 5 - Value of a is ", a)

a**=9
print ("Line 6 - Value of a is ", a)
print("Check", 3*3*3*3*3*3*3*3*3)

a//=9
print ("Line 7 - Value of a is ", a)
print ("Check:", 19683//9)
```

```
Line 1 - Value of a is  30
Line 2 - Value of a is  21
Line 3 - Value of a is  189
Line 4 - Value of a is  21.0
Line 5 - Value of a is  3.0
Line 6 - Value of a is  19683.0
Check 19683
Line 7 - Value of a is  2187.0
Check: 2187
```

In [23]:

```
#a=9
a+=9
print(a)
```

2196.0

2196.0

COMPARISON OPERATORS

These operators are used to compare two values:

Operator	Name	Action
==	Equal to	x == y
!=	Not Equal to	x != y
>	Greater Than	x > y
<	Less Than	x < y
>=	Greater Than Equal to	x >= y
<=	Less Than Equal to	x <= y

In [24]:

```
x=9
y=13

x==y
print("Is x == y?", x==y)

x!=y
print("Is x!=y?", x!=y)

x>y
print("Is x>y?", x>y)

x<y
print("Is x<y?", x<y)

x>=y
print("Is x>=y?", x>=y)

x<=y
print("Is x<=y?", x<=y)
```

```
Is x == y? False
Is x!=y? True
Is x>y? False
Is x<y? True
Is x>=y? False
Is x<=y? True
```

LOGICAL OPERATORS

These operators are used to compare two values:

Operator	Details	Example
and	TRUE is both the statements are CORRECT	A<5 and B>10
or	TRUE if one of the statement is CORRECT	A<5 or B>10
not	REVERSE of the RESULT	not(A<5 and B>10)

In [25]:

```
x=110
y=200
z=300

x>150 or y>150
```

Out[25]:

```
True
```

In [26]:

```
x>=100 and y>150
```

Out[26]:

```
True
```

In [27]:

```
x>=100 and y>150
```

Out[27]:

```
True
```

In [28]:

```
not (x>=100 and y>150)    #REVERSING THE RESULT BY USING NOT
```

Out[28]:

```
False
```

IDENTITY OPERATORS

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location

Operator	Details	Example
is	TRUE if both variables are actually the same object	`A is B`
is not	TRUE if both variables are actually NOT the same object	`A is not B`

In [29]:

```
x=100
y=100
```

In [30]:

```
x is y
```

Out[30]:

```
True
```

In [31]:

```
x=300
y=300
```

In [32]:

```
x is y
```

Out[32]:

```
False
```

In [33]:

```
x=-4
y=-4
```

In [34]:

```
x is y
```

Out[34]:

```
True
```

MEMORY STRUCTURE IN PYTHON

Before we proceed further, we must understand the memory structure in python and how variables and values are stored in python.

Normally, when you assign a value to a variable, you can think of a bag/box a(variable) and putting a value 1 in it.

Now, when you change the value of a to 2, it simply reassigns the new value to it.

When we assign one variable to another variable, it simply makes two boxes with the same value.

Assignment: a=b

But in Python variables work more like tags unlike the boxes you have seen before. When you do an assignment in Python, it tags the value with the variable name.

For example: When you have variables a, b, c having a value 10, it doesn't mean that there will be 3 copy of 10s in memory. There will be only one 10 and all the variables a, b, c will point to this value.

In [35]:

```
a=10
b=10
c=10
id(a), id(b), id(c)
```

Out[35]:

```
(1819766064, 1819766064, 1819766064)
```

In [36]:

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

Out[36]:

```
1819766096
```

`id()` will return an objects memory address (object's identity). As you have noticed, when you assign the same integer value to the variables, we see the same ids. But this assumption does not hold true all the time. See the following for example:

In [37]:

```
x=500
y=500
```

```
print(id(x) , id(y))

x is y
```

```
1770429252592 1770429254000
```

Out[37]:

```
False
```

Even after assigning the same integer values to different variable names, we are getting two different ids here. Python implementation keeps an array of integer objects for all integers between -5 and 256. So, when we create an integer in that range, they simply back reference to the existing object

In [38]:

```
s1='hello'
s2='hello'
id(s1), id(s2)
```

Out[38]:

```
(1770429443856, 1770429443856)
```

In [39]:

```
s1==s2
```

Out[39]:

```
True
```

In [40]:

```
s1 is s2
```

Out[40]:

```
True
```

In [41]:

```
s3='hello world!'
s4='hello world!'
s3==s4
```

Out[41]:

```
True
```

In [42]:

```
s3 is s4
```

Out[42]:

```
False
```

When the string was a simple and shorter one the variable names where referring to the same object in memory. But when they became bigger, this was not the case. This is called **INTERNING**, and Python does interning (to some extent) of shorter string literals (as in s1 and s2) which are created at compile time. But in general, Python string literals creates a new string object each time (as in s3 and s4). Interning is runtime dependant and is always a trade-off between memory use and the cost of checking if you are creating the same string.

MEMBERSHIP OPERATORS

These operators are used to test if a sequence is presented in an object

Operator	Details	Example
in	TRUE if a value/sequence is present in an object	`A in B`
not in	TRUE if a value/sequence is NOT present in an object	`A not in B`

To be covered in detail along with lists, tuples, & dictionaries

COMMENTS IN PYTHON

To add a comment in any cell, we use the **#** button and then write the comment

HEADING: `ESC+1`,`ESC+2`, `ESC+3`, `ESC+4`, `ESC+5`, `ESC+6`

MARKDOWN: `ESC+M`

RAW NBCONVERT: `ESC+R`

Other Keyboard Shortcuts:

https://datalya.com/blog/python-data-science/jupyter-notebook-command-mode-keyboard-shortcuts

STRING FUNCTIONS

STRIP()

The `strip()` method removes any whitespace from the beginning or the end:

In [43]:

```
A = "  Python is a language"
print(A.strip())
print(A)
```

```
Python is a language
  Python is a language
```

LEN()

The `len()` method returns the length of a string:

In [44]:

```
A = "Python is a language"
print(len(A))
```

```
20
```

In [45]:

```
x = "This is a class"
print(len(x))
```

```
15
```

In [46]:

```
len(A)
```

Out[46]:

```
20
```

In [47]:

```
A.upper()
```

Out[47]:

```
'PYTHON IS A LANGUAGE'
```

In [48]:

```
A
```

Out[48]:

```
'Python is a language'
```

Count()

The `count()` function returns the count of a character in a string:

In [49]:

```
A = "Python is a language"
print(A.count("g"))
```

```
2
```

LOWER()

The `lower()` method returns the string in lower case:

In [50]:

```
A = "Python is a language"
print(A.lower())
```

```
python is a language
```

UPPER()

The `upper()` method returns the string in upper case:

In [51]:

```
A = "Python is a language"
print(A.upper())
```

```
PYTHON IS A LANGUAGE
```

In [52]:

```
A=input("Enter your name ")
```

```
Enter your name Ankur
```

In [53]:

```
print(type(A))
```

```
<class 'str'>
```

In [54]:

```
A = input("Enter Number")
```

```
Enter Number13
```

In [55]:

```
type(A)
```

Out[55]:

```
str
```

In [56]:

```
x = float(input("Enter Number"))
```

```
Enter Number14
```

In [57]:

```
print(x)
print(type(x))
```

```
14.0
<class 'float'>
```

TITLE()

The `title()` is used to make the first letter of each word in uppercase

In [58]:

```
A = "Python is a language"
print(A.title())
```

```
Python Is A Language
```

FIND()

Find the index of the first appearing substring within a string using `str.find()`

In [59]:

```
A[0]
```

Out[59]:

```
'P'
```

In [60]:

```
A[5]
```

Out[60]:

'n'

In [61]:

```
A[0:4]     #[start_index, stop_index] inclusive, exclusive
```

Out[61]:

'Pyth'

In [62]:

```
A[10]
```

Out[62]:

'a'

In [63]:

```
A.find("z")
```

Out[63]:

-1

If the substring does not appear, find() returns -1:

In [64]:

```
A.find("L")
```

Out[64]:

-1

REPLACE()

The `replace()` method replaces a string with another string:

In [65]:

```
A = "Python is a language"
print(A.replace("a", "z"))
```

```
Python is z lznguzge
```

SPLIT()

The `split()` method splits the string into substrings and forms a list

In [66]:

```
##  Splitting each word into a separate element in a list:
A = "Python is a language"
print(A.split())
```

```
['Python', 'is', 'a', 'language']
```

In [67]:

```
## Splitting by any other character in the string:
A = "Python is a language"
```

```
A = "Python is a language"
print(A.split("a"))
```

```
['Python is ', ' l', 'ngu', 'ge']
```

CONCATENATE

You can join (concatenate) two strings with the plus `(+)` operator:

In [68]:

```
"Python" + " is a language"
```

Out[68]:

```
'Python is a language'
```

In [69]:

```
String = "Hello People"

print(String + " This is a lovely morning")
```

```
Hello People This is a lovely morning
```

For complex string operations of this sort is preferable to use the **`str.format()`** function. str.format() takes in a template string with curly braces as placeholders for values you provide to the function as the arguments. The arguments are then filled into the appropriate placeholders in the string:

Str.format()

In [70]:

```
name = "Jack"
age = 10
city = "Paris"

String1 = "My name is {} I am {} and I live in {}"
String1.format(name, age, city)
```

Out[70]:

```
'My name is Jack I am 10 and I live in Paris'
```

STRING INPUT

Python allows for command line `input`. That means we can ask the user for input

In [71]:

```
x=input("Enter your name:")
print("Welcome,", x)
```

```
Enter your name:Ankur
Welcome, Ankur
```