BASIC RULES FOR PYTHON VARIABLES:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters (A-z, 0-9) and underscores
- Variable names are case-sensitive, e.g., amount, Amount and AMOUNT are three different variables.
- A variable name cannot be any of the Python keywords.

Remember that variable names are case-sensitive

Camel Case

Each word, except the first, starts with a capital letter:

```
myVariableName = “John”
```

Pascal Case

Each word starts with a capital letter:

```
MyVariableName = "John"
```

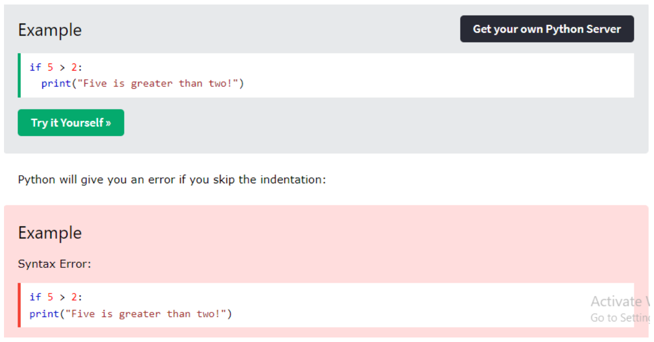
Snake Case

Each word is separated by an underscore character:

```
my_variable_name = "John"
```

PYTHON INDENTATION

Indentation refers to the spaces at the beginning of a code line.
Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
Python uses indentation to indicate a block of code.



Example

```
if 5 > 2:
  print("Five is greater than two!")
```

Python will give you an error if you skip the indentation:

Example

Syntax Error:

```
if 5 > 2:
print("Five is greater than two!")
```

CASTING

If you want to specify the data type of a variable, this can be done with casting

```
x = int(1.4)
y= float("3")
z = str(2)

print(x)
print(y)
print(z)

1, 3.0, 2
```

GET THE TYPE

```
x = 1
y = 2.8
z = 3 + 2j

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

<class 'int'>
<class 'float'>
<class 'complex'>
```

	Data Type	Example
Text Type	str	“Hello World”
Numeric Types	int	10
	float	10.5
	complex	1j
Sequence Types	list	["apple", "banana", "orange"]
	tuple	("apple", "banana", "orange")
	range	range(5)
Mapping Type	dict	{"name" : "vishnu", "age" : 27}
Set Types	set	{"apple", "banana", "orange"}
	frozenset	frozenset({"apple", "banana", "orange"})
Boolean Type	bool	True, False
Binary Types	bytes	b"Hello"
	bytearray	bytearray(5)
	memoryview	memoryview(bytes(5))

PYTHON VARIABLES - ASSIGN MULTIPLE VALUES

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
```

```
Orange
Banana
Cherry
```

```
x = y = z = "Orange"

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

```
Orange
Orange
Orange
```

Unpack a Collection

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

```
fruits = ["apple", "banana", "cherry"]
x, y, z = fruits
print(x)
print(y)
print(z)
```

```
apple
banana
cherry
```

```
x = "Python"
y = "is"
z = "awesome"
print(x, y, z)
```

Python is awesome

```
x = "Python "
y = "is "
z = "awesome"
print(x + y + z)
```

Python is awesome

```
x = 5
y = 10
print(x + y)

OUTPUT :15
---------------------------------------------------------
x = 5
y = "John"
print(x + y)
```

TypeError: unsupported operand type(s) for +: 'int' and 'str

GLOBAL VARIABLES

```
x = "awesome"
def myfunc():
  print("Python is " + x)
myfunc()
```

Python is awesome

```
x = "awesome"
def myfunc():
  x = "fantastic"
  print("Python is " + x)
myfunc()
print("Python is " + x)
```

Python is fantastic
Python is awesome

```
def myfunc():
  global x
  x = "fantastic"
myfunc()
print("Python is " + x)
```

```
Python is fantastic
```

```
x = "awesome"
def myfunc():
  global x
  x = "fantastic"
myfunc( )
print("Python is " + x)
```

```
Python is fantastic
```

The variable **x** is initially set to **"awesome".**
Inside **the myfunc() function, the global keyword is used to indicate that the function will modify the global variable x** instead of creating a new local variable.
When **myfunc()** is called, it changes the value of **x to "fantastic".**
Finally, the print statement concatenates **"Python is "** with the updated global value of x, which is now **"fantastic"**

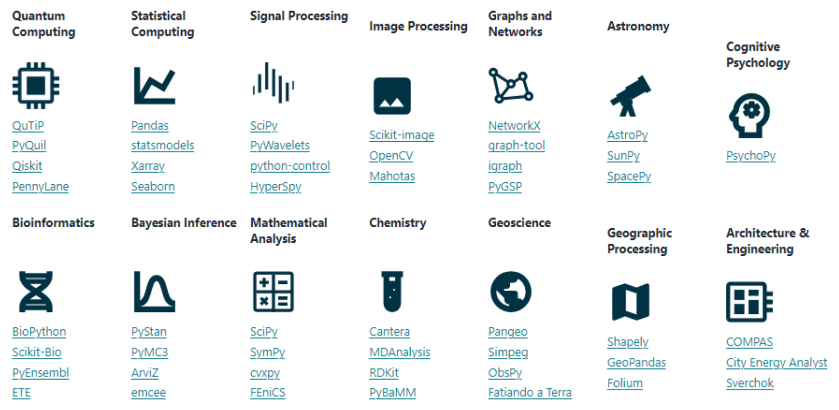
```
x = "awesome"

def myfunc():
  global x
  print("Python is " + x)   # Print "awesome" before changing it
  x = "fantastic"

myfunc()

print("Python is " + x)


Python is awesome
Python is fantastic
```

String Input

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

```
import random
print(random.randrange(1, 10))
```

NumPy - NumPy is the fundamental package for scientific computing with Python



Quantum Computing	Statistical Computing	Signal Processing	Image Processing	Graphs and Networks	Astronomy	Cognitive Psychology
QuTiP	Pandas	SciPy	Scikit-image	NetworkX	AstroPy	PsychoPy
PyQuil	statsmodels	PyWavelets	OpenCV	graph-tool	SunPy	
Qiskit	Xarray	python-control	Mahotas	igraph	SpacePy	
PennyLane	Seaborn	HyperSpy		PyGSP		

Bioinformatics	Bayesian Inference	Mathematical Analysis	Chemistry	Geoscience	Geographic Processing	Architecture & Engineering
BioPython	PyStan	SciPy	Cantera	Pangeo	Shapely	COMPAS
Scikit-Bio	PyMC3	SymPy	MDAnalysis	Simpeg	GeoPandas	City Energy Analyst
PyEnsembl	ArviZ	cvxpy	RDKit	ObsPy	Folium	Sverchok
ETE	emcee	FEniCS	PyBaMM	Fatiando a Terra		

• **SciPy - SciPy is** a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering

Matplotlib

- Matplotlib is a Python 2D plotting library

example: https://matplotlib.org/stable/plot_types/index.html