General syntax

- **Python language is Case Sensitive**

```
bob=100
Bob=90
print(bob)
print(Bob)
```

- **Comment line starts with #**

```
# this is a comment line
```

- **Naming variables**
  - A variable name can start with a letter or underscore
  - All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
  - must not contain spaces or special characters ! @ # % ^ & *
  - must not be any keyword defined in the language, or reserved names: in, for, etc.

```
1v = 2       #error, it starts with a number
v_# = 2      #error, has special character
for="Hello" #error, for is a reserved word

v1 = 2     #variable assignment can have spaces
v1=2
print(v1)
```

Data types

In Python, data are classified in types.

In Python, data take the form of *objects (abstract data types),* and Python programs manipulate these objects

An object's type defines the possible values and operations that type supports.

In this course we will study these built-in types:

- Numeric: Integer, Float
- Sequence: String, List, Tuple
- Mapping: Dictionary

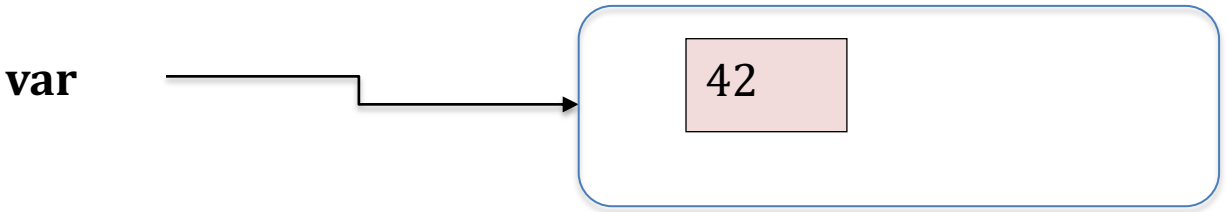
Example	**Data types**
`i = 42`	`int`
`f = 20.5`	`float`
`s = "pop"`	`str`
`L = ["apple", "banana", "cherry"]`	`list`
`T = ("apple", "banana", "cherry")`	`tuple`
`D = {"name" : "John", "age" : 36}`	`dict`

Data: Value, type and identity

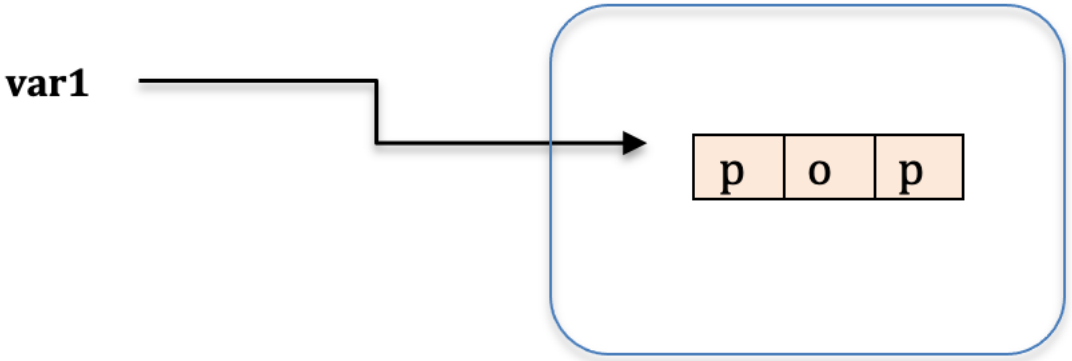
Every object (data type) has these three attributes:

- **Value** stored by the object.
- **Type:** the kind of object that is created. integer, list, string etc. ***type()* function**
- **Identity:** the address that the object refers to in the computer's memory. ***id()* function**

```
var = 42
print(var)
type(var)
id(var)
```




```
var1='pop'
print(var1)
type(var1)
id(var1)
```

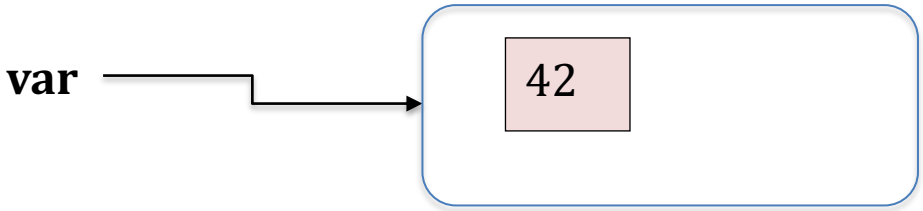


Variables and Object References

What happens when you make a variable assignment in Python?

```
var = 42 #integer object is created and is bound to the name var
type(var)
<class 'int'>
```

An assignment binds the name of a variable to the value stored in the computer memory



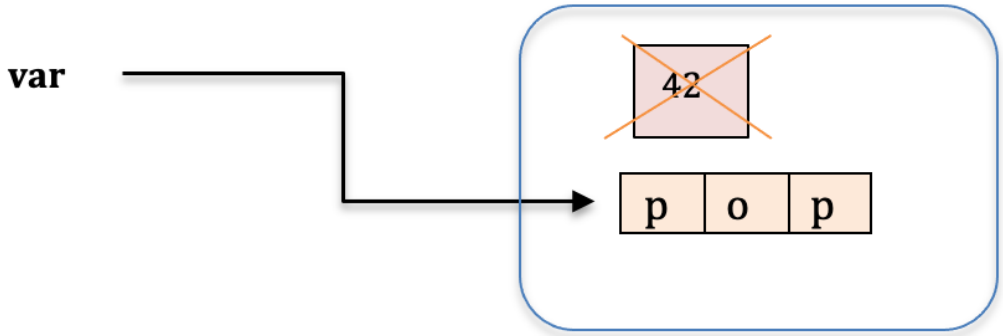
You can check the memory address of a variable by using the function *id( )*

```
id(var)
140266786844240
```

Variables and Object References

Now we change the value of variable var

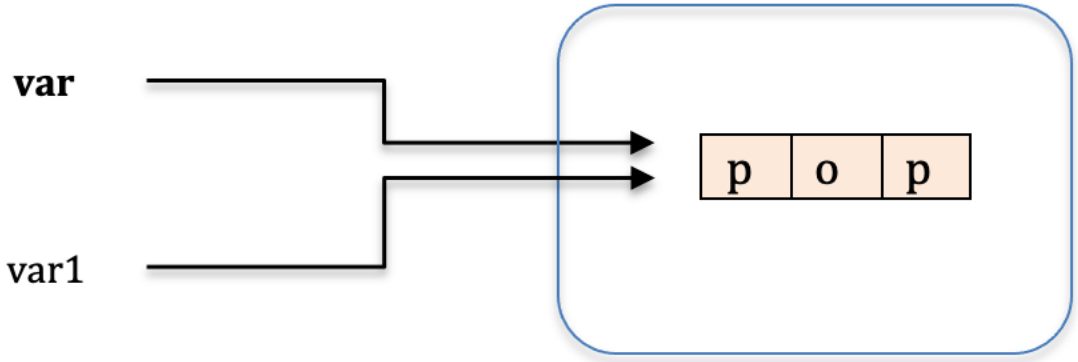
```
var = "pop"  #string object is created and is bound to the name var
type(var)
<class 'str'>

id(var)
140266786848496
#compare with previous id
```



- **Python variables are references to objects, but the actual data is contained in the objects.**
- Python uses dynamic binding for variables.
- In Python, the data type of a variable can change during program execution.

Variables and Object References


```
var="pop"
var1 = var
id(var)
id(var1)
#id is the same
```



Both variables reference the same object.

In this case, Python will not make a copy of the object (value) but will make a new reference to the same value.

This is important to know when we will talk about mutable objects!

Multiple Assignment

Can assign a single value to several variables simultaneously

```
a = b = c = 1
```

Can assign different values to different variables in one line. This is called tuple unpacking.

```
a,b,c = 1,"john",[1,2,3,4]
```

Scalar objects: Numeric type

Scalar objects - cannot be subdivided.

- int represents integer, ex 5
- float represents real number, ex 3.45
- Bool represents Boolean values True and False
- NoneType has only one value None – it represents the absence of a type

In Python, numeric data type represents the data which has numeric value

```
var=42
print(var)
print(type(var))
```


```
var1=3.14
print(var1)
print(type(var1))
```

Operations with Numeric types

Expressions - You can combine objects and operators to form expressions.

`<object> <operator> <object>` → `expression evaluates to a value`

```
i=8
j=2
type(i+j)

f=1.2
type(f*j)
```

Object Operator Object	result
int + int int – int int * int int**int	int
int/int	float
int + float int – float int * float int/float int**float float**int	float
float + float float – float float * float float/float float**float	float

```
z=i/j      #result is stored in variable z
type(z)
i % j      #the reminder of a division
```

Operations with Numeric types

Operators and order of increasing precedence:

Left → Right flow

`+`	**addition**
`-`	**subtraction**
`*`	**multiplication**
`/`	**division**
`**`	**power**
`()`	**parentheses**

Typical precedence you might expect in math

```
3*2**3      # order of precedence results in 24

(3*2)**3    # order of precedence results in 216
```

number syntax and scientific notation

Try these

```
m1=100,100,100
m2=100100100
```

What is the data type of m1 and m2?

Scientific notation is **a way of writing very large or very small numbers**.
Example: 650000000.0 in scientific notation is 6.5×10^8

```
6.5*10**8   #650000000.0
6.5*pow(10,8) #650000000.0
```

math functions - Here some built-in math functions

```
abs(-4)      #absolute value
pow(2,3)     #power
```

The math module

If you need to perform more advanced operations, such as exponential, logarithmic, trigonometric, or power functions, you can import and use the **math module**

```
help("math") #at the IPython Console to enter documentation page of a module
https://docs.python.org/3/library/math.html #online documentation
```

Generic import. If you want to import all functions and constants available in the math module

```
import math
math.log(100)    #specify the module name followed by the function name
math.pi          #specify the module name followed by the constant name

import math as m  #you can rename the module name in your code.
m.log(x)
m.pi
```

Function import: If you want to only import specific functions and/or constants

```
from math import log, sqrt, pi
log(100)          #specify only the function name
pi                #specify only the constant name
```

Type conversion: int() and float()

Sometimes it is necessary to convert values from one type to another.

For example, you cannot sum a string with an integer:

```
s='1210'
i=1340
print(i+s)
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

TypeError message reported for *unsupported operand type(s)* because we wanted to add integer data type to a string data type.

We will use the int() function to convert a string type to a numeric type, and then add the values.

```
s1=int(s)
print(i+s1)

print(i+int(s))
```

Type conversion: int() and float()

Here some examples on int() and float() functions

```
int('2014')  #create integer object 2014 from string  '2014'
2014

int(3.141592) #create integer object 3 from float 3.141592
3

float('1.99') #create float object 1.99 from string  '1.99'
1.99

float(5)      #create float object 5.0 from integer 5
5.0
```

print function


```
print(arg1,arg2,..,argN)
```

- Each argument is a data type.
- The print function will make one space between output arguments.
- Arithmetic operations can be performed within the print function.

```
x=2
y=5
D={1:10,2:20}
L=[1,3,6]
s="Hello"

print(x, y, x/y, x**y)
2 5 0.4 32

print("Hello", x+y, "times")  #text is a string type
Hello 7 times

print(D,L,x**y,s)
{1: 10, 2: 20} [1, 3, 6] 32 Hello
```

Tab and newline characters in print function

```
"\n"   #newline character is a string type
"\t"   #tab character is a string type
```

Try them in a print statement:

```
print("Hello", x+y, "\ntimes")
Hello 100
times

print("Hello", x+y, "\n","times")
Hello 100
 times


print("Hello\ntimes\ttimes")
Hello
times times

print("Hello","\n","times","\t","times")
Hello
 times  times
```

print function and formatting operator %

To specify the format, follow the same syntax as printf() in awk and bash – the difference is the syntax of listing arguments

```
print("format1"  %arg1)
print('format1 format2'  %(arg1,arg2))

%[width].[precision]type

%type
%s  string
%d  integer
%e  scientific notation
%f  real number

print('%.1f' %12.345)
print('%10.2f %.1e' %(12.345, 100000.0))

x=5
y=15.253
print('%.2f' %y)
print('%d %.2f' %(x, y/x))
print("Hi, my name is %.5s and I have %d brother." %("MariaG", 1))
```

Input Function

Input function reads a line from standard input, converts it to a string, and returns that string.
Input function is used to make interactive scripts

Make another script called **my-input.py** and in it use the input function to ask the user to enter a number

```
number=input("Enter a number: ")
```

- print what's stored in the variable number
- print the data type of the variable number
- use the variable number to multiply number by 2
- print the result of the multiplication to screen

Run the script
Did you perform arithmetic multiplication or something else?

Now modify the script in order to output the arithmetic operation

Input Function

Input function reads a line from standard input, converts it to a string, and returns that string.
Input function is used to make interactive scripts.

Make another script called **my-input.py** and in it use the input function to ask the user to enter a two numbers separated by a comma

```
number=input("Enter two numbers separated by comma: ")
```

- print what's stored in the variable number
- print the data type of the variable number
- use type conversion to sum the two numbers