Python Kickstart: Launching Your Coding Adventure

Dr. Garima Jaiswal
Bennett University

Agenda

Python: Introduction

Python: Basics

Python: Flow control structures and loops

Python: List, Tuple, Dictionary

Strings

Functions & Recursion

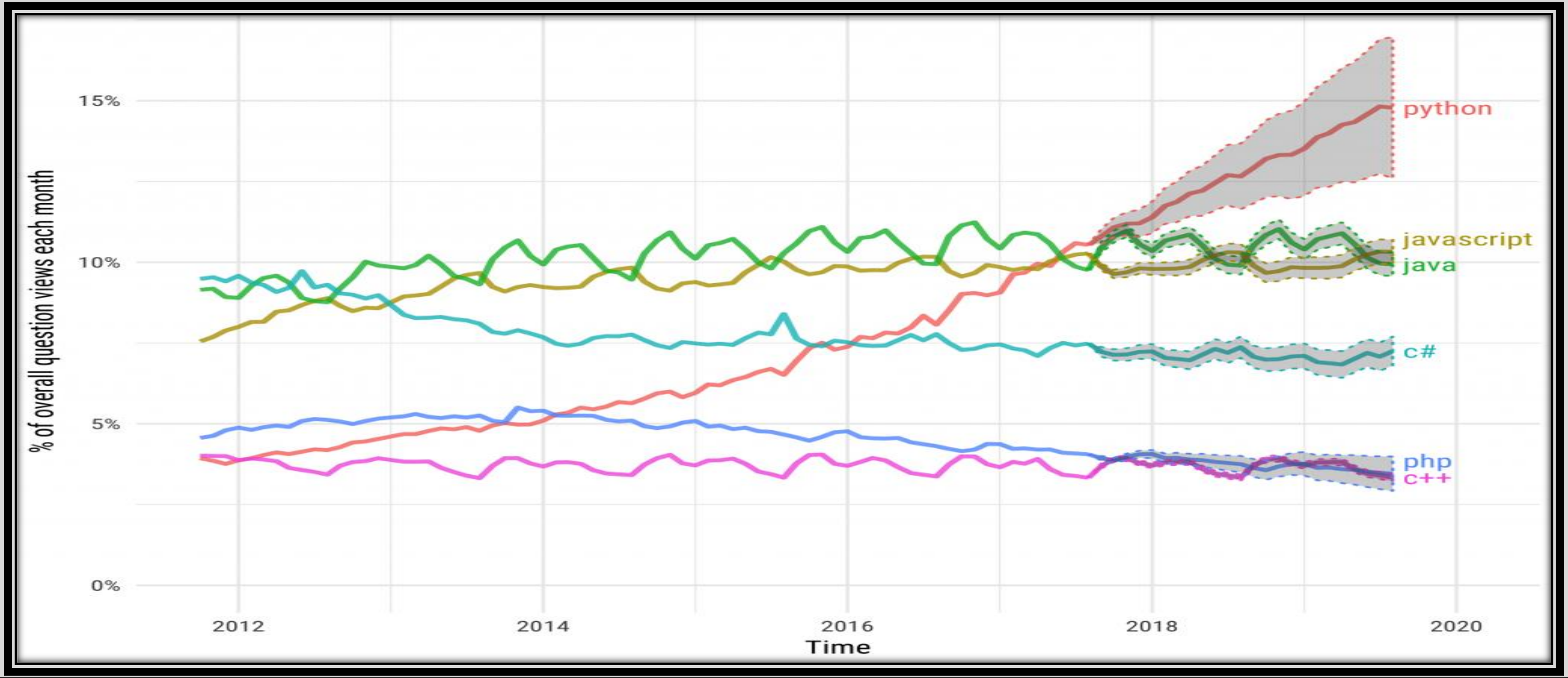
Overview of Python libraries

Introduction

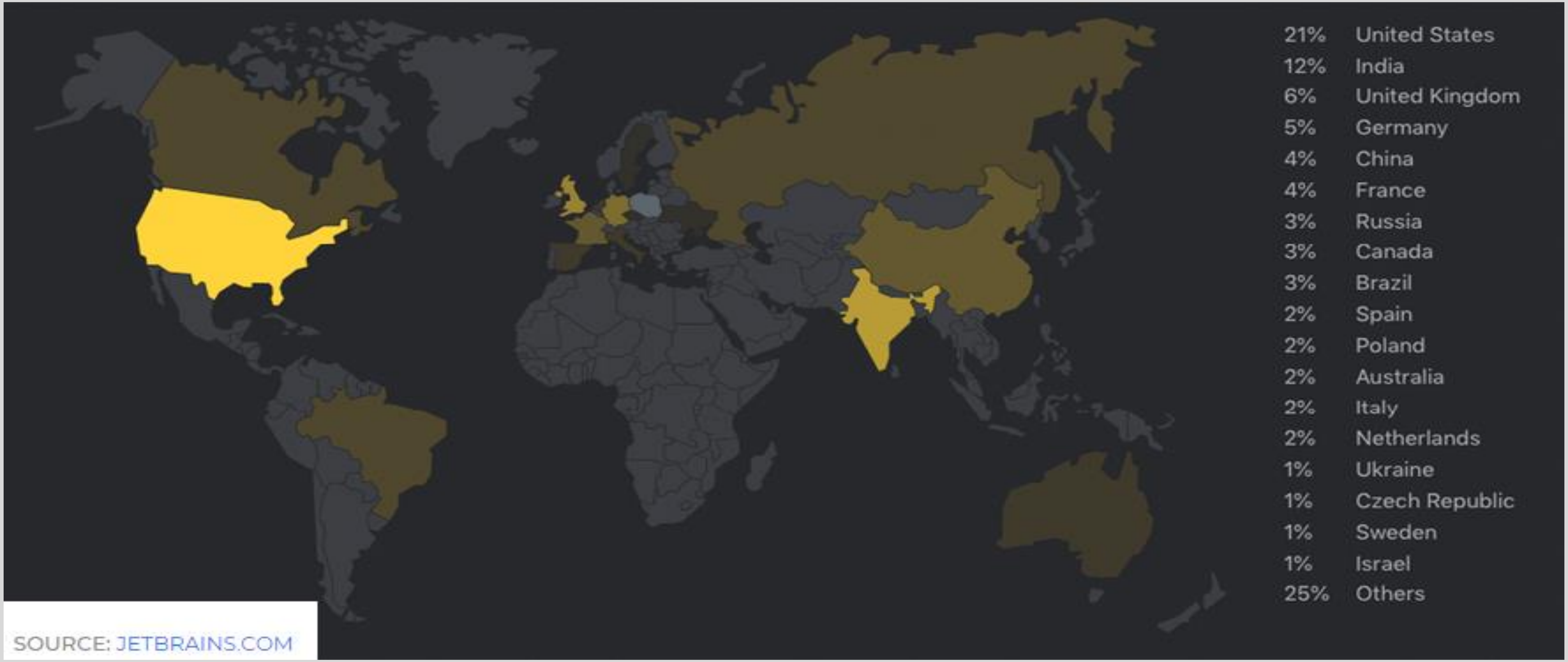
- Python is a popular programming language.
- It was created by Guido van Rossum, and released in 1991.
- Inspired by his favorite show's (Flying Circus) creator Monty Python.
- High level, Interpreted language with easy syntax and dynamic semantics.

"high-level programming language, and its core design philosophy is all about code readability and a syntax which allows programmers to express concepts in a few lines of code".

Projection of future traffic for major programming languages



Python Web Development Company



21%	United States
12%	India
6%	United Kingdom
5%	Germany
4%	China
4%	France
3%	Russia
3%	Canada
3%	Brazil
2%	Spain
2%	Poland
2%	Australia
2%	Italy
2%	Netherlands
1%	Ukraine
1%	Czech Republic
1%	Sweden
1%	Israel
25%	Others

SOURCE: JETBRAINS.COM

WHY Python

01 Simplicity

02 Large Community

03 High Demand-Supply Ratio

04 Large Number of Frameworks

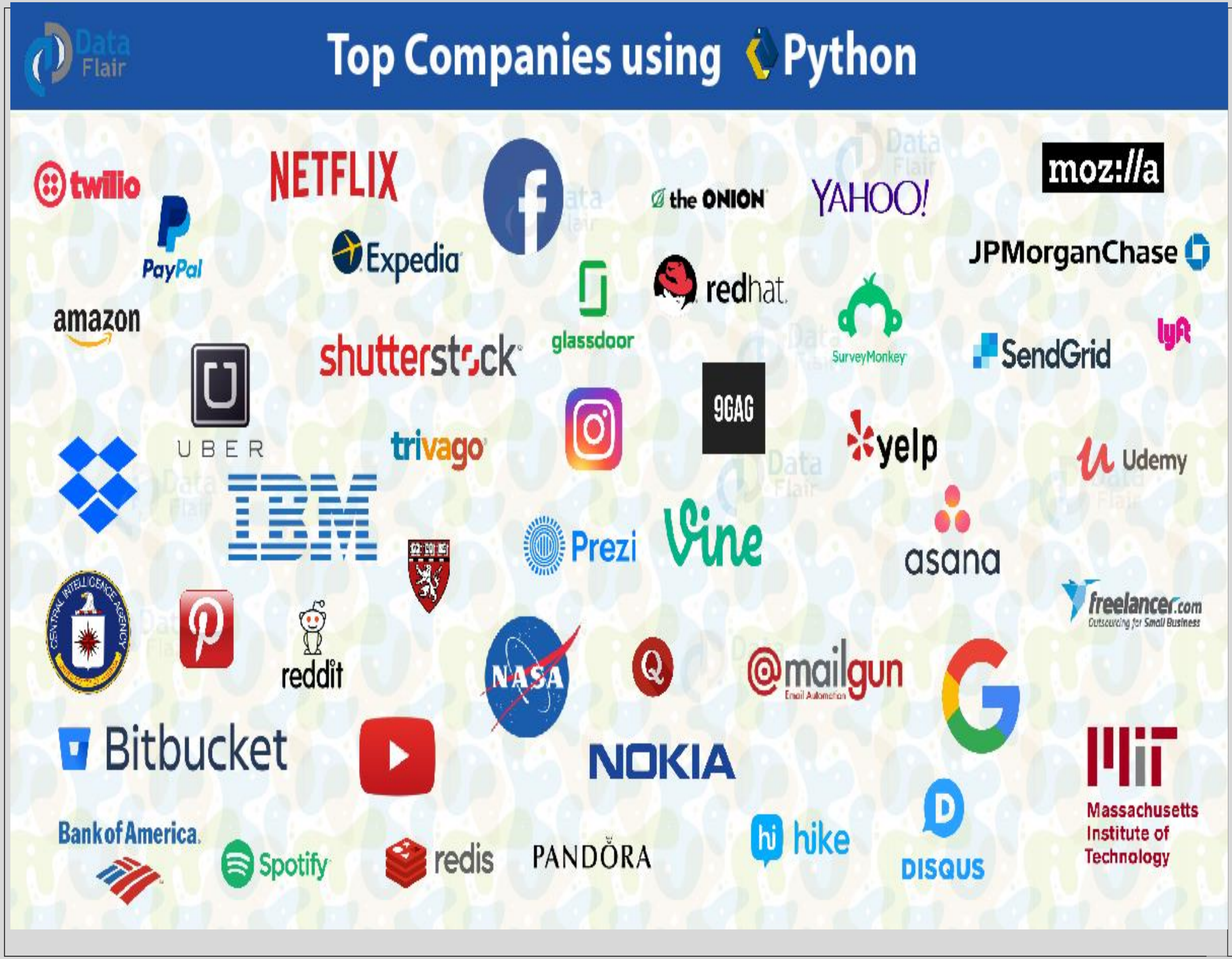
05 Chosen Language for AI and ML

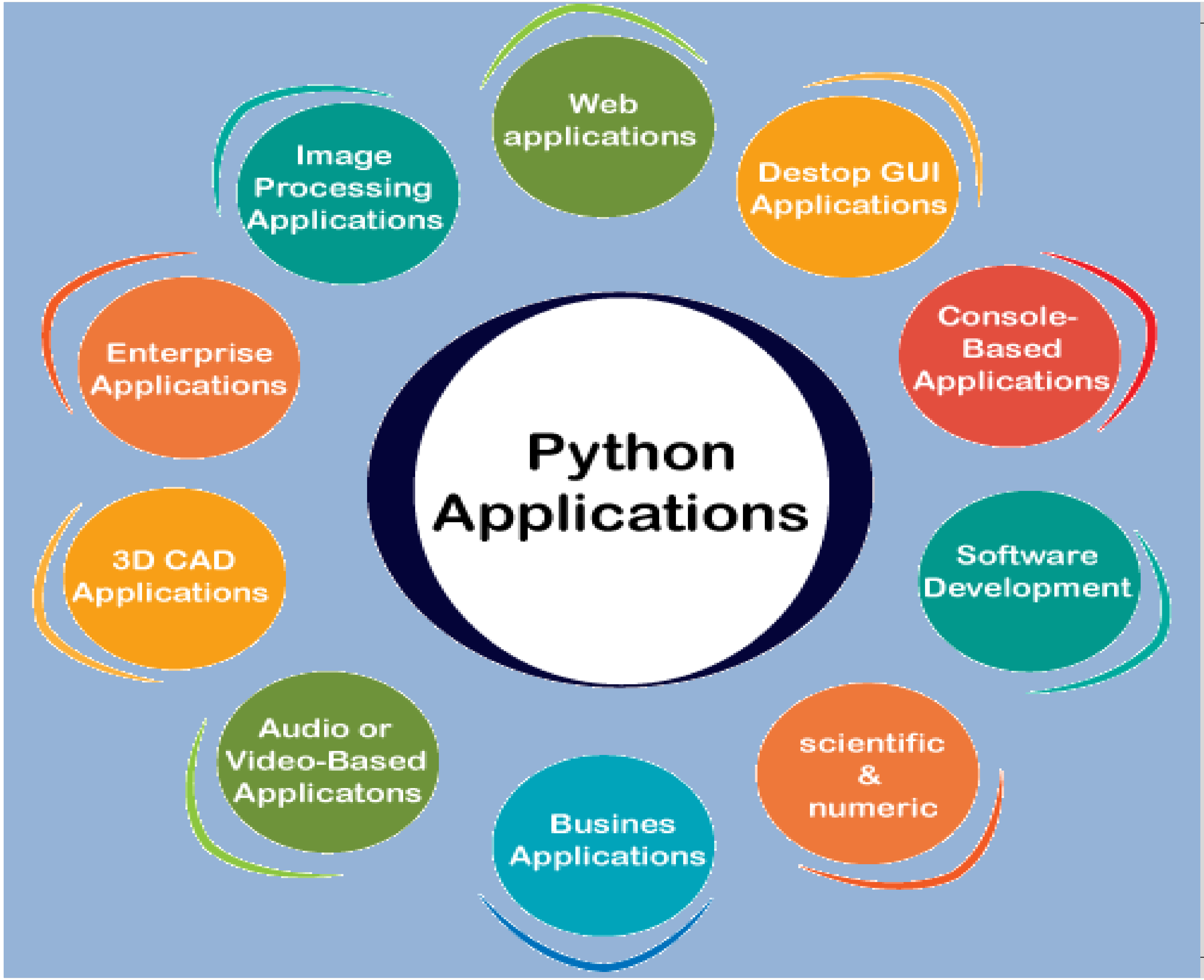
06 Make your own DIYs

Features of Python?

- □ Simplicity
- □ Open source
- □ Portability
- □ Embeddable and Extensible
- □ Interpreted
- □ Huge Libraries
- □ Object Orientation

Who Uses Python?





Applications

- Web Development : Django, Flask
- Game Development: PySoy :3D game engine ,PyGame: library for game development. Games such as Civilization-IV, Disney's Toon town Online, Vega Strike etc. have been built using Python.
- Machine Learning and Artificial Intelligence : Pandas, Scikit-Learn, NumPy
- Data Science and Data Visualization

BASIC CONCEPTS OF PYTHON

Expression

- Expressions consist of ***values*** (such as 2) and ***operators*** *(such as +), and they can always evaluate (that is,* reduce) down to a single value.
- >>> 2 + 2

4

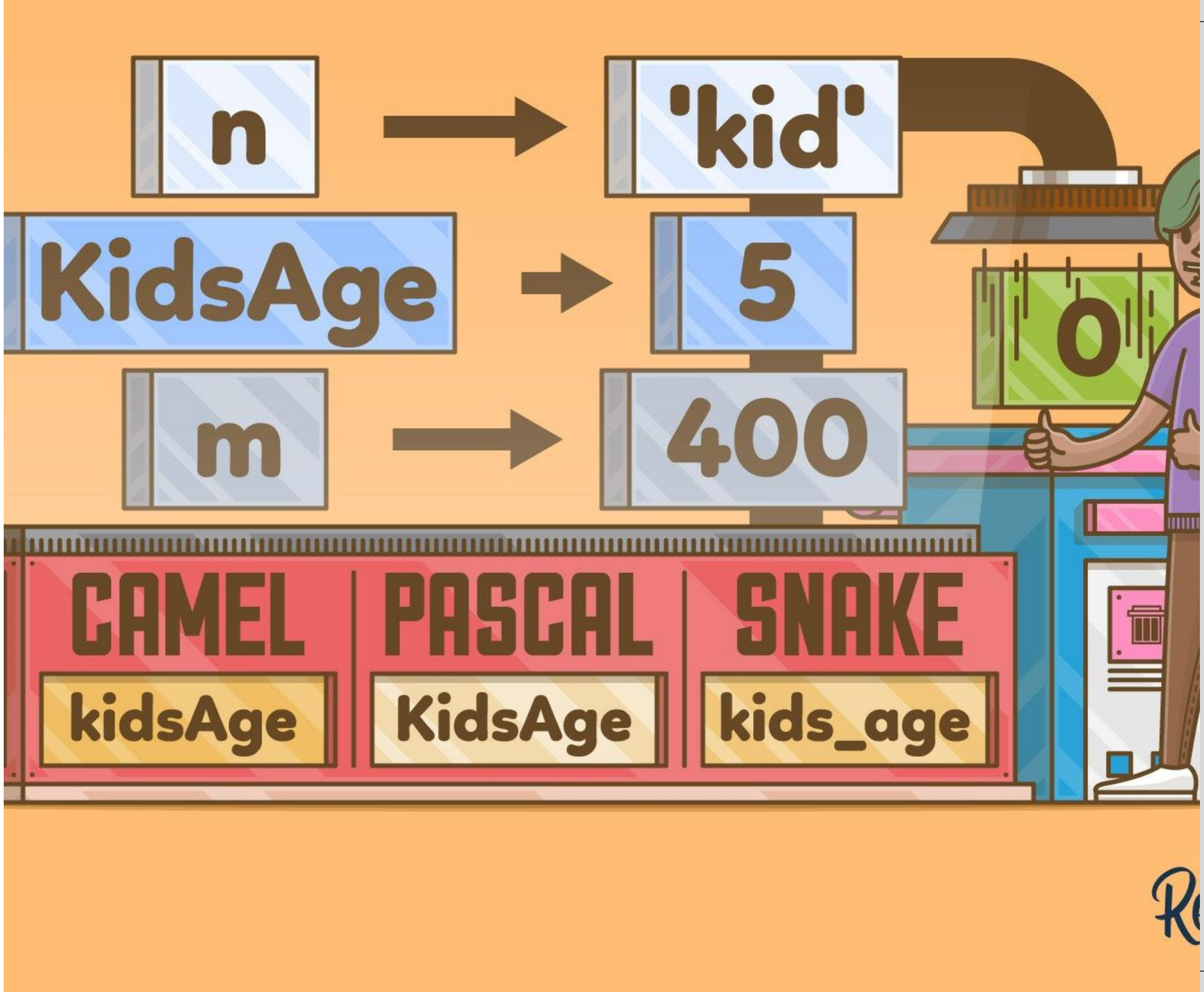
Order of Operations

Operator	Operation	Example	Evaluates to…
**	Exponent	2 ** 3	8
%	Modulus/remainder	22 % 8	6
//	Integer division/floored quotient	22//8	2
/	Division	22 / 8	2.75
*	Multiplication	3*5	15
-	Subtraction	5-2	3
+	Addition	2+2	4

Note: Order of precedence: **,(*,/,//,% L to R), (+,- L to R)

Cont..

```
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565878 * 578453
28093077826734
>>> 2 ** 8
256
>>> 23 / 7
3.2857142857142856
>>> 23 // 7
3
>>> 23 % 7
2
>>> 2 + 2
4
>>> (5 - 1) * ((7 + 1) / (3 - 1))
16.0
```

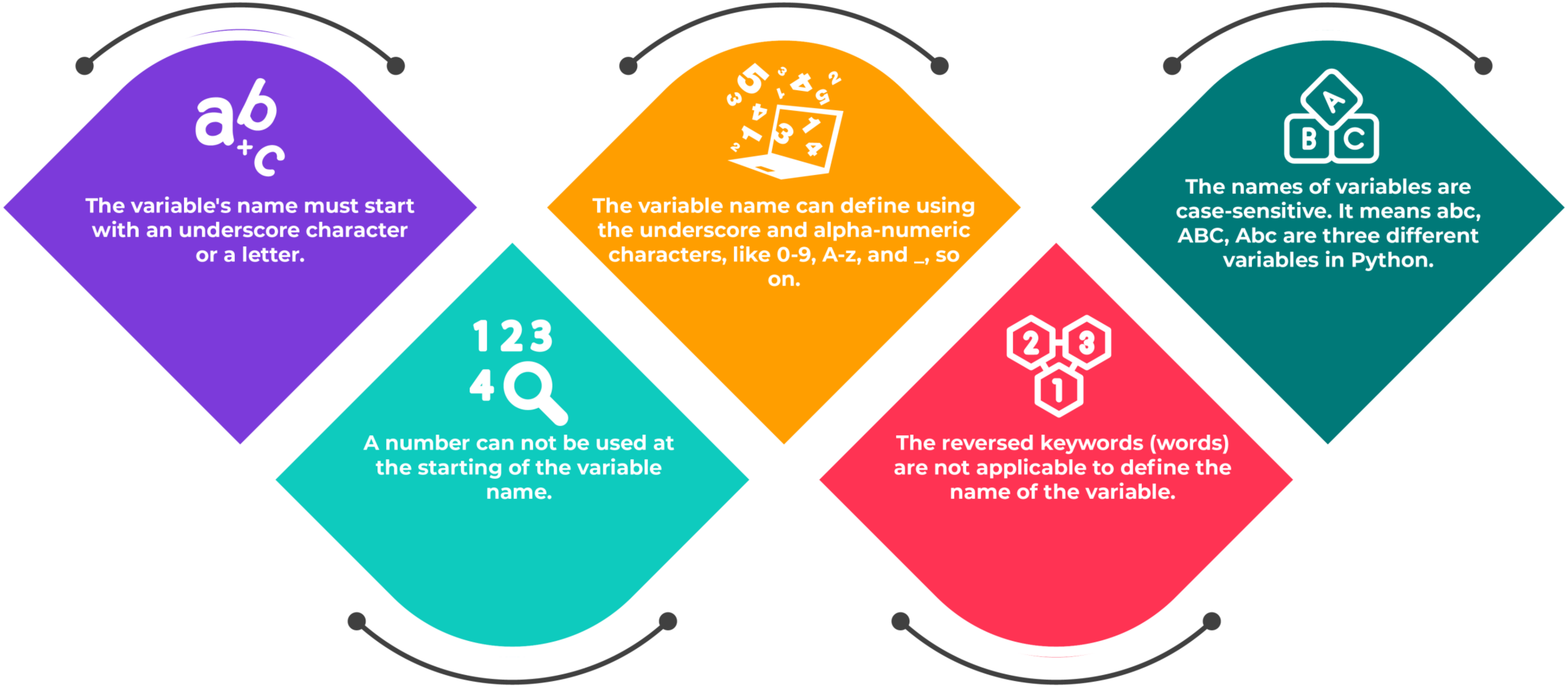


Variables

- ❑ reserved memory locations to store values.
- ❑ based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- ❑ Creating variables;
- ❑ Casting ;
- ❑ Type Function;
- ❑ Updating;
- ❑ Multiple Assignment;

Is there any rule for creating variables in Python?

Yes, there are certain rules that users need to follow for creating a variable in Python. And the rules are as follows:



The variable's name must start with an underscore character or a letter.

A number can not be used at the starting of the variable name.

The variable name can define using the underscore and alpha-numeric characters, like 0-9, A-z, and _, so on.

The reversed keywords (words) are not applicable to define the name of the variable.

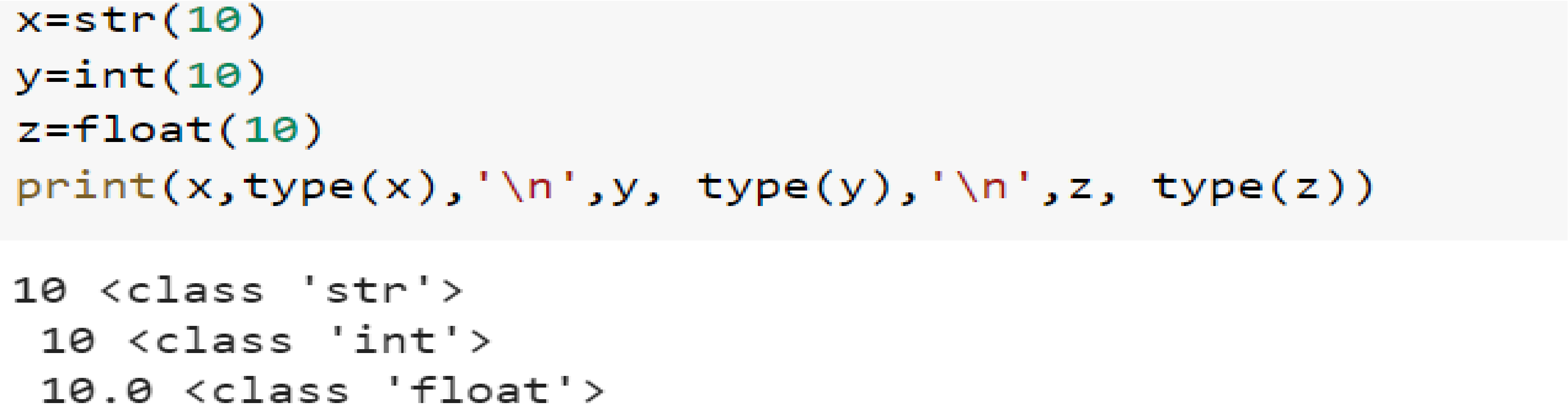
The names of variables are case-sensitive. It means abc, ABC, Abc are three different variables in Python.

Cont…

```
>>> spam = 40
>>> spam
40
>>> eggs = 2
>>> spam + eggs
42
>>> spam + eggs + spam
82
 >>> spam = spam + 2
>>> spam
42
```

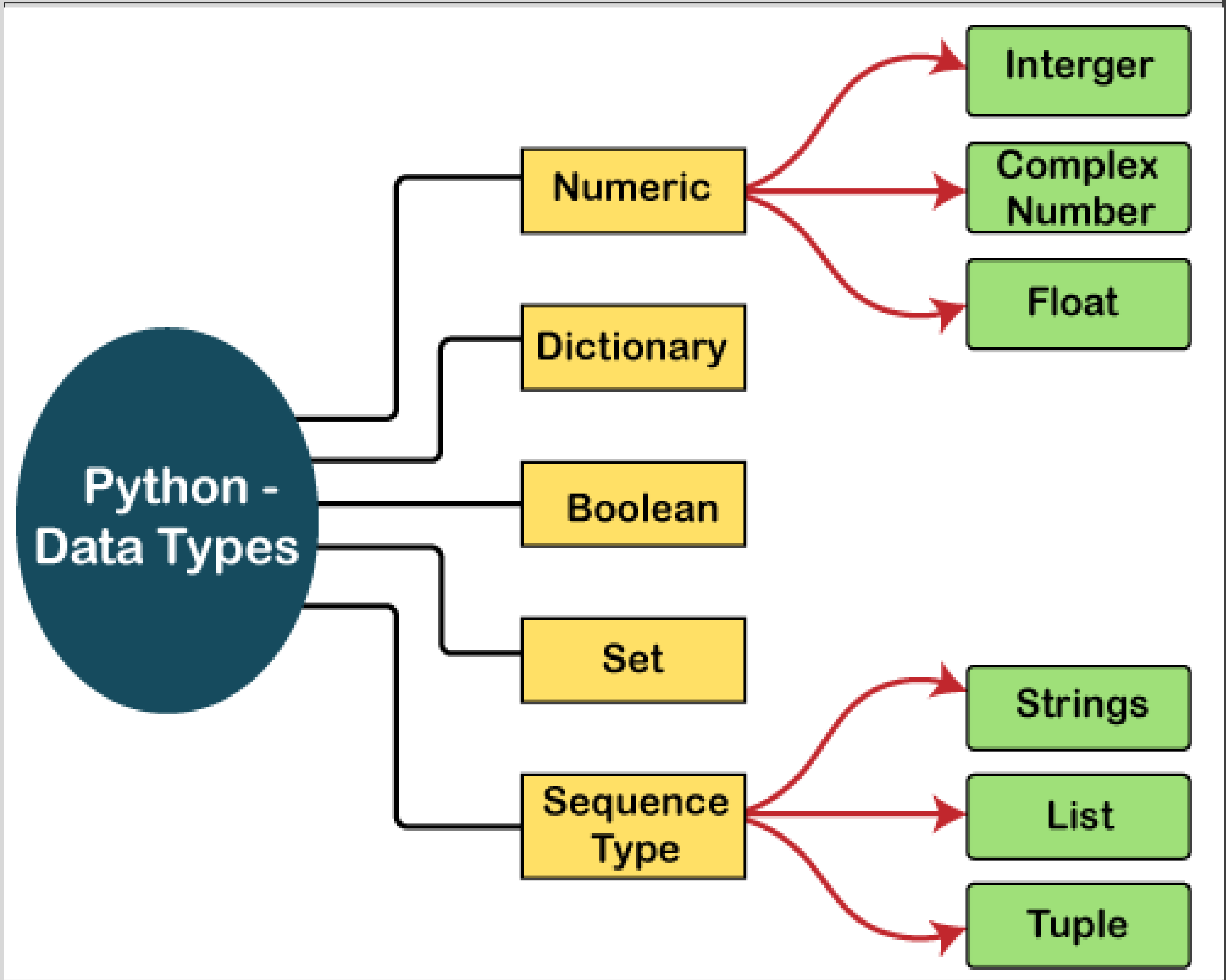
Python Data Types

- Variables can hold values, and every value has a data-type.
- Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it.
- The interpreter implicitly binds the value with its type.



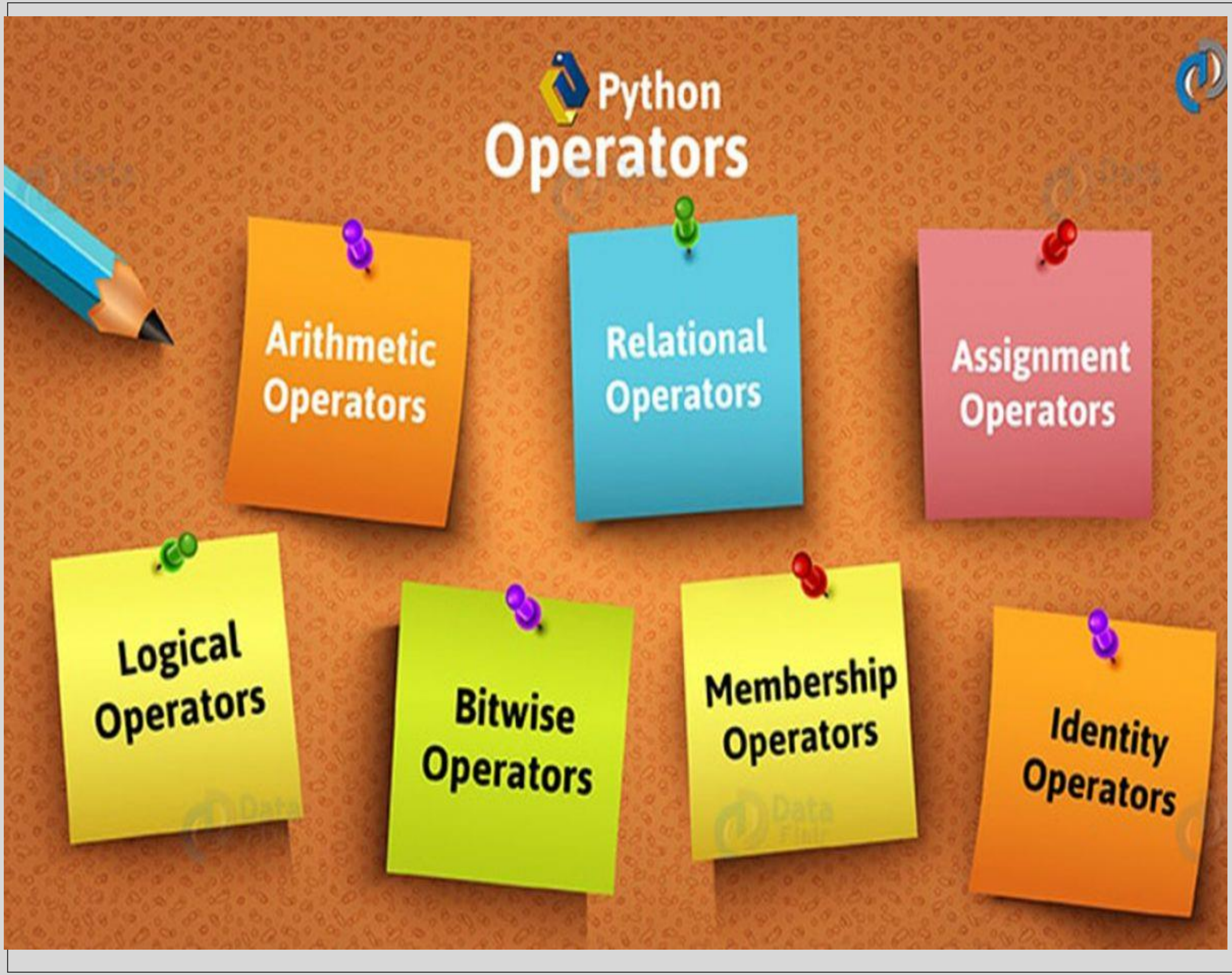
```python
x=str(10)
y=int(10)
z=float(10)
print(x,type(x),'\n',y, type(y),'\n',z, type(z))
```

```
10 <class 'str'>
 10 <class 'int'>
 10.0 <class 'float'>
```



Data Types

- Int - Integer value can be any length. Python has no restriction on the length of an integer. Its value belongs to int
- Float - Float is used to store floating-point numbers. It is accurate upto 15 decimal points.
- complex - A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively.



Operators

- The operator can be defined as a symbol which is responsible for a particular operation between two operands.
- Operators are the pillars of a program on which the logic is built in a specific programming language

Boolean Values

- *Boolean data type has only two values:* True and False.
- >>> **spam = True**
- >>> **spam**
- True

Comparison Operators

- == Equal to
- != Not equal to
- < Less than
- > Greater than
- <= Less than or equal to
- >= Greater than or equal to

Boolean Operators

- The three Boolean operators (and, or, and not) are used to compare Boolean values.

The str(), int(), and float() Functions

- The str(), int(), and float() functions will evaluate to the string, integer and floating-point forms of the value you pass, respectively.

FLOW CONTROL

Flow Control Statements

1) IF Statements

```
if name == 'Alice':
    print('Hi, Alice.')
```

2)ELSE Statements

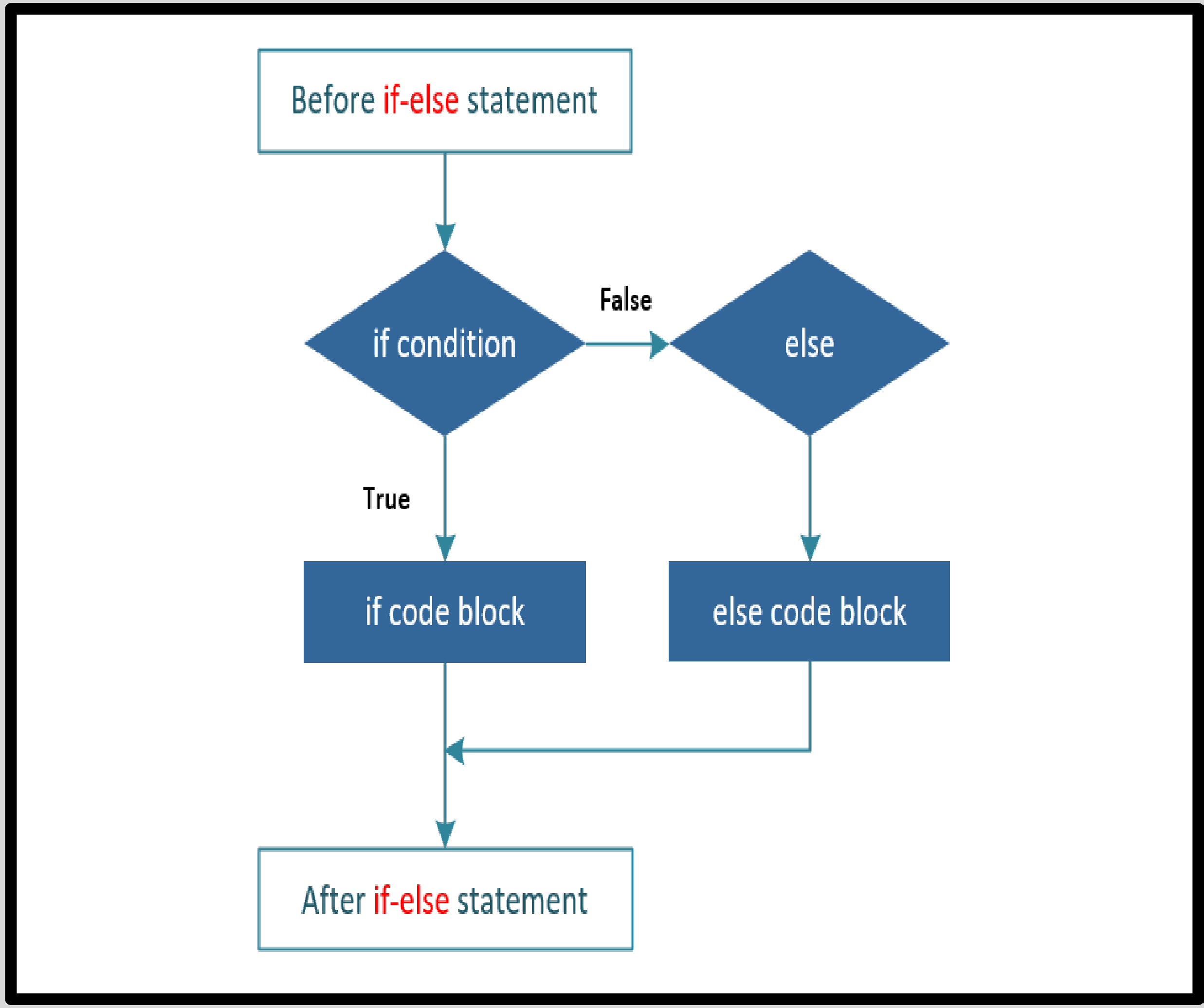
```
if name == 'Alice':
   print('Hi, Alice.')
else:
       print('Hello, stranger.')
```

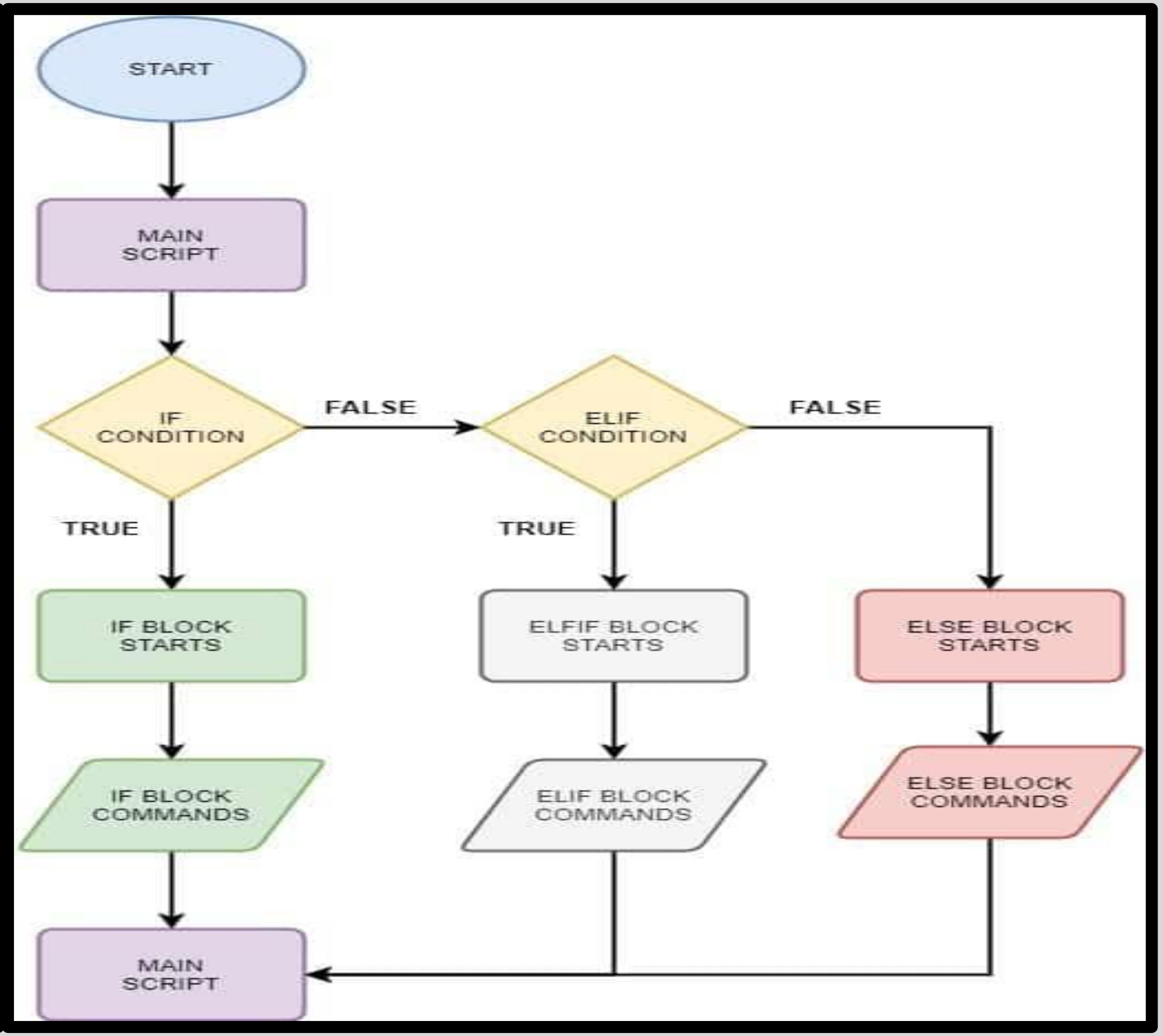
Cont..

3)ELIF Statements

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')
elif age > 100:
    print('You are not Alice, grannie.')
```

Conditional Statements





while Loop Statements

```
spam = 0
while spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

Output???

```
spam = 0
if spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

for Loops and the range() Function

- The for keyword
- A variable name
- The in keyword
- A call to the range() method with up to three integers passed to it
- A colon
- Starting on the next line, an indented block of code (called the for clause)

```
for i in range(5):
    print(‘ Hi')
```

The Starting, Stopping, and Stepping Arguments to range()

```
for i in range(12, 16):
        print(i)
for i in range(0, 10, 2):
        print(i)
for i in range(5, -1, -1):
         print(i)
```

Try out…

Write a short program that prints the numbers 1 to 10 using a for loop. Then write an equivalent program that prints the numbers 1 to 10 using a while loop.

LISTS

Lists

- A *list is a value that contains multiple values in an ordered sequence*.
- Values inside the list are also called *items*.

```
>>> [1, 2, 3]
[1, 2, 3]
>>> ['cat', 'bat', 'rat', 'elephant']
['cat', 'bat', 'rat', 'elephant']
>>> ['hello', 3.1415, True, None, 42]
['hello', 3.1415, True, None, 42]
 >>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam
['cat', 'bat', 'rat', 'elephant']
```

Getting Individual Values in a List with Indexes

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']

>>> spam[0]

'cat'

>>> spam[1]

'bat‘

>>>'Hello ' + spam[0]

'Hello cat‘

>>> spam[10000]

IndexError: list index out of range

>>> spam[1.0]

TypeError: list indices must be integers, not float
```

Cont…

```
spam = [['cat', 'bat'], [10, 20, 30, 40, 50]]

>>> spam[0]
['cat', 'bat’]

>>> spam[0][1]
'bat’

>>> spam[1][4]
50
```

Negative Indexes

```
spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[-1]
'elephant'
>>> spam[-3]
'bat'
```

Getting Sublists with Slices

- index can get a single value from a list,
- a *slice can get several values* from a list, in the form of a new list.

```
spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0:4]
['cat', 'bat', 'rat', 'elephant']
>>> spam[1:3]
['bat', 'rat']
>>> spam[0:-1]
['cat', 'bat', 'rat']
>>> spam[:2]
['cat', 'bat']
>>> spam[1:]
['bat', 'rat', 'elephant']
>>> spam[:]
['cat', 'bat', 'rat', 'elephant']
```

Getting a List’s Length with len()

```
>>> spam = ['cat', 'dog', 'moose']
>>> len(spam)
3
```

Changing Values in a List with Indexes

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1] = 'aardvark'
>>> spam
['cat', 'aardvark', 'rat', 'elephant']
>>> spam[2] = spam[1]
>>> spam
['cat', 'aardvark', 'aardvark', 'elephant']
```

List Concatenation and List Replication

```
>>> [1, 2, 3] + ['A', 'B', 'C']
[1, 2, 3, 'A', 'B', 'C']
>>> ['X', 'Y', 'Z'] * 3
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']
```

Removing Values from Lists with del Statements

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat', 'elephant']
```

Using for Loops with Lists

```
for i in [0, 1, 2, 3]:
    print(i)
```

Output:

```
0
1
2
3
```

In and not in Operators

```
>>> 'howdy' in ['hello', 'hi', 'howdy', 'heyas']
True
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> 'cat' in spam
False
```

Finding a Value in a List with the index() Method

```
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> spam.index('hello')
0
>>> spam.index('heyas')
3
>>> spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']
>>> spam.index('Pooka')
1
```

Adding Values to Lists with the append() and insert() Methods

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.append('moose')
>>> spam
['cat', 'dog', 'bat', 'moose']

>>> spam = ['cat', 'dog', 'bat']
>>> spam.insert(1, 'chicken')
>>> spam
['cat', 'chicken', 'dog', 'bat']
```

- **Note:** The append() and insert() methods are list methods and can be called only on list values, not on other values such as strings or integers.

Removing Values from Lists with remove()

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('bat')
>>> spam
['cat', 'rat', 'elephant']
```

- Attempting to delete a value that does not exist in the list will result in a ValueError error.
- If the value appears multiple times in the list, only the first instance of the value will be removed.

Sorting the Values in a List with the sort() Method

```
>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
>>> spam.sort()
>>> spam
['ants', 'badgers', 'cats', 'dogs', 'elephants']
>>> spam.sort(reverse=True)
>>> spam
['elephants', 'dogs', 'cats', 'badgers', 'ants']
```

Cont…

1)The sort() method sorts the list in place; don’t try to capture the return value by writing code like spam = spam.sort().

2)You cannot sort lists that have both number values *and string* values in them.

3) sort() uses “ASCIIbetical order” rather than actual alphabetical order for sorting strings. This means **uppercase letters come before lowercase letters.**

Cont..

```
>>> spam = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats']
>>> spam.sort()
>>> spam
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats']
```

To sort the values in regular alphabetical order

```
>>> spam = ['a', 'z', 'A', 'Z']
>>> spam.sort(key=str.lower)
>>> spam
['a', 'A', 'z', 'Z']
```

This causes the sort() function to treat all the items in the list as if they were lowercase without actually changing the values in the list.

Mutable and Immutable Data Types

- A list value is a *mutable* data type: It can have values added, removed, or changed.
- String is *immutable*: *It cannot be changed.*

The Tuple Data Type

- immutable.
- typed with parentheses ( )

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[0]
'hello'
>>> eggs[1:3]
(42, 0.5)
>>> len(eggs)
3
```

Converting Types with the list() and tuple() Functions

```
>>> tuple(['cat', 'dog', 5])
('cat', 'dog', 5)
>>> list(('cat', 'dog', 5))
['cat', 'dog', 5]
>>> list('hello')
['h', 'e', 'l', 'l', 'o']
```

References

```
>>> spam = 42
>>> cheese = spam
>>> spam = 100
>>> spam
100
>>> cheese
42
```

Cont…

```
>>> spam = [0, 1, 2, 3, 4, 5]
*>>> cheese = spam
 >>> cheese[1] = 'Hello!'
>>> spam
[0, 'Hello!', 2, 3, 4, 5]
>>> cheese
[0, 'Hello!', 2, 3, 4, 5]
```

- **Note:**

*copies only the list reference in spam to cheese, not the list value itself. This means the values stored in spam and cheese now both refer to the same list.

- So when you modify the first element of cheese ,you are modifying the same list that spam refers to.

DICTIONARIES

Dictionary

- *Dictionary is a collection of many values*.
- **indexes** for dictionaries can use many different data types, not just integers.
- **Indexes** for dictionaries are called ***keys****, and a* ***key*** *with its associated* **value** is called a ***key-value pair***.
- Dictionary is typed with braces**, {}.**
- myCat = {'size': 'fat', 'color': 'gray', disposition': 'loud'}
- Dictionary’s keys :'size', 'color', and 'disposition'.
- Values for these keys : 'fat', 'gray‘, 'loud’.

```
>>> myCat['size']
'fat‘
```

Dictionaries vs. Lists

List	Dictionary
first item in a list named spam would be spam[0].	no “first” item in a dictionary
order of items matters for determining whether two lists are the same	it does not matter in what order the key-value pairs are typed in a dictionary.

>>> **spam = ['cats', 'dogs', 'moose']**
>>> bacon = ['dogs', 'moose', 'cats']
>>> **spam == bacon**
False
	>>> **eggs = {'name': 'Zophie', 'species': 'cat', 'age': '8'}**
	>>> **ham = {'species': 'cat', 'age': '8', 'name': 'Zophie'}**
	>>> **eggs == ham**
	True

The keys(), values(), and items() Methods

- Will return list-like values.
- The values returned by these methods are not true lists: They cannot be modified and do not have an append() method.
- spam = {'color': 'red', 'age': 42}

spam.keys()	dict_keys(['color', 'age'])
Spam.values()	dict_values(['red', 42])
Spam.items	dict_items([('color', 'red'), ('age', 42)])

Checking Whether a Key or Value Exists in a Dictionary

```
>>> spam = {'name': 'Zophie', 'age': 7}
>>> 'name' in spam.keys()
True
>>> 'Zophie' in spam.values()
True
>>> 'color' in spam.keys()
False
```

The get() Method

- Tedious task: to check whether a key exists in a dictionary before accessing that key's value.
- get() method that takes two arguments:
- ➢ key of the value to retrieve
- ➢ fallback value to return if that key does not exist.

Cont…

```
spam = {'color': 'red', 'age': 42}
print(spam.get('color',0))
print(spam.get('size',0))
print(spam('size'))
```

Output:

```
red
0
TypeError: 'dict' object is not callable
```

The setdefault() Method

- To set a value in a dictionary for a certain key only if that **key does not already have a value.**

```
spam = {'name': 'Pooka', 'age': 5}
if 'color' not in spam:
    spam['color'] = 'black‘
```

Cont…

- The setdefault() method offers a way to do this in one line of code.
- The first argument :key to check for,
- The second argument: is the value to set at that key if the key does not exist.
- If the key does exist, the setdefault() method returns the key’s value.

Cont…

```
spam = {'color': 'red', 'age': 42}
spam.setdefault('size',0)
print(spam)
```

Output: {'color': 'red', 'age': 42, 'size': 0}

```
spam = {'color': 'red', 'age': 42}
spam.setdefault('age',10)
print(spam)
```

0utput: {'color': 'red', 'age': 42}

STRINGS

COMPOUND DATA TYPE

- Strings are qualitatively different from Integer and Float type.
- Characters which collectively form a String is a Compound Data Type.

For Eg.

```
fruit = “apple”
letter = fruit[1]
print (letter)
Output  : p          // (index value starts from 0 as in C & C++)
```

LENGTH OF STRINGS

- The inbuilt function to find the length of a string is **‘len()’.**

  For Eg.

  ```
  fruit = “banana”
  len(fruit)
  ```

  Output : 6

- To get the last letter we might try

  ```
  length = len(fruit)
  last = fruit[length]                    #ERROR
  ```

  (because there is no character at 6th place)

LENGTH OF STRING (to be continued…..)

- Right Method to do this is :

  length = **len(fruit)**

  last = fruit[length-1]

- Another way to get the elements from last is :

  fruit[-1]   # yields the last letter

  fruit[-2] # yields the second last letter

TRAVERSAL USING WHILE LOOP

- Processing one character at one time.

For Eg.

```
index = 0
while index < len(fruit):
    letter = fruit[index]
    print (letter)
    index = index + 1
```

(Take care of the indentation)

TRAVERSAL USING FOR LOOP

- For loop provides us a privilege to access the characters without using index.

    For Eg.

```
fruit=“apple”
for char in fruit:
        print (char)
```

(Each time through the loop a character is assigned to the variable char)

STRING SLICES

- A segment of a string is called a slice, i.e. a character.
- The syntax to select a slice from a string is **a[n:m]**, where a contains strings, n is the starting index and m is the end index.
- Includes the first index and excluding the last index.

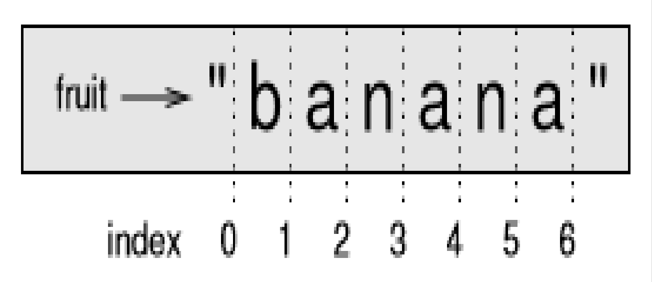
Eg:

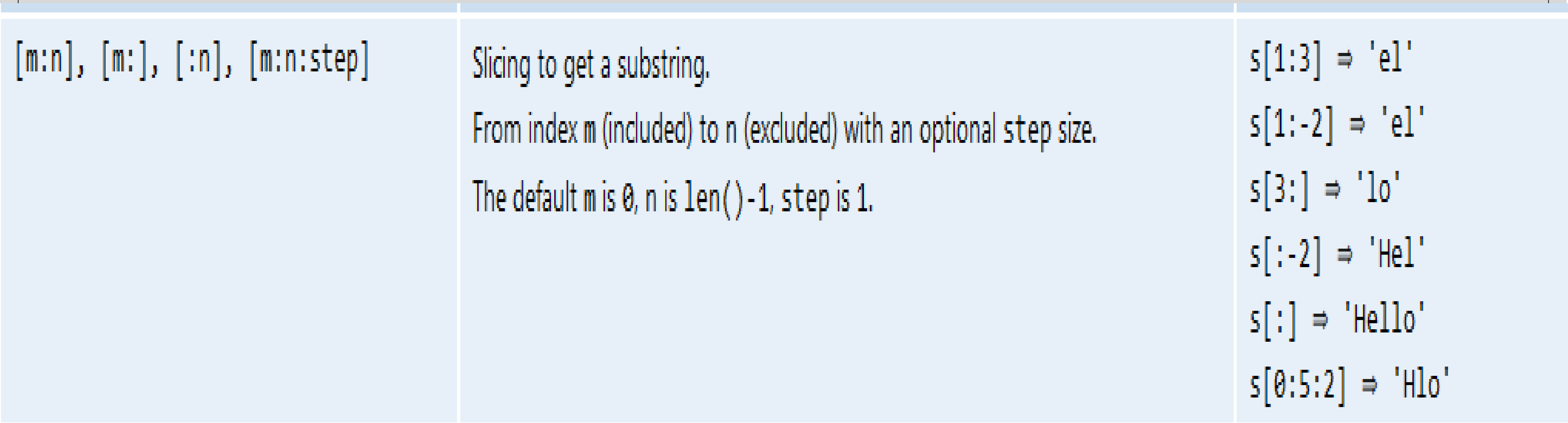
```
s= “Peter, Paul, and Mary”
 print(s [0:5])                # Peter
 print(s [6:10])               # Paul
print(s [15:21])               # Mary
```

STRING SLICES

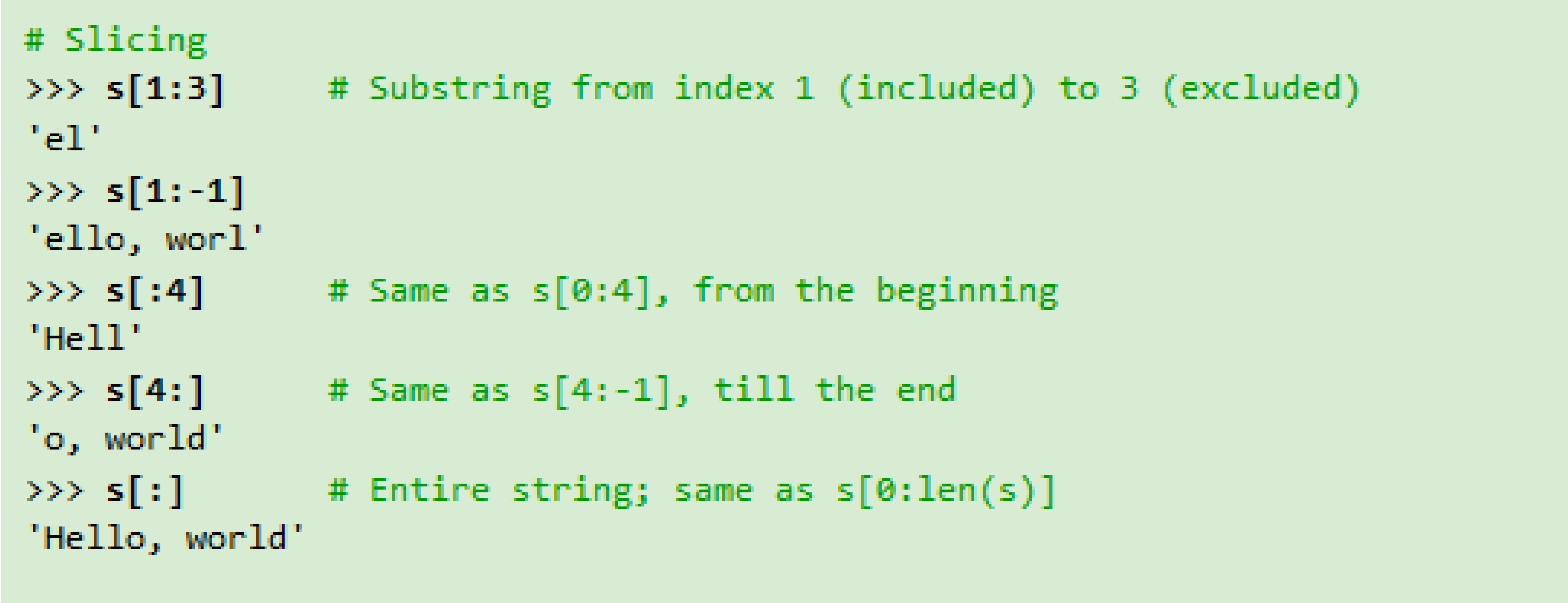
(to be continued…..)

```
fruit = “banana”
fruit[ : 3]                    #ban
fruit[ 3 :]                    #ana
fruit[:]              ?
```





[m:n], [m:], [:n], [m:n:step]	Slicing to get a substring. From index m (included) to n (excluded) with an optional step size. The default m is 0, n is len()-1, step is 1.	s[1:3] ⇒ 'el' s[1:-2] ⇒ 'el' s[3:] ⇒ 'lo' s[:-2] ⇒ 'Hel' s[:] ⇒ 'Hello' s[0:5:2] ⇒ 'Hlo'



```
# Slicing
>>> s[1:3]     # Substring from index 1 (included) to 3 (excluded)
'el'
>>> s[1:-1]
'ello, worl'
>>> s[:4]      # Same as s[0:4], from the beginning
'Hell'
>>> s[4:]      # Same as s[4:-1], till the end
'o, world'
>>> s[:]       # Entire string; same as s[0:len(s)]
'Hello, world'
```

STRINGS ARE IMMUTABLE

- An existing string cannot be modified.

For Eg :

```
greeting = "Hello, world!“
greeting[0] = ’J’                          # ERROR!


print (greeting)
```

Output : Hello, world

STRINGS ARE IMMUTABLE

(to be continued….)

- The Solution of the problem is

```
greeting = "Hello, world!“
newGreeting = ’J’ + greeting[1:]
print (newGreeting)
```

Output : Jello, World

- The original string remains intact.

Python String find() Method

- Definition and Usage
  - The find() method finds the first occurrence of the specified value.
  - The find() method returns -1 if the value is not found.

- Syntax
  - *string*.find(*value, start, end*)
  - *value* Required. The value to search for
  - *Start* Optional. Where to start the search. Default is 0
  - *End* Optional. Where to end the search. Default is to the end of the string

STRING MODULE **(to be continued….)**

Find Function :

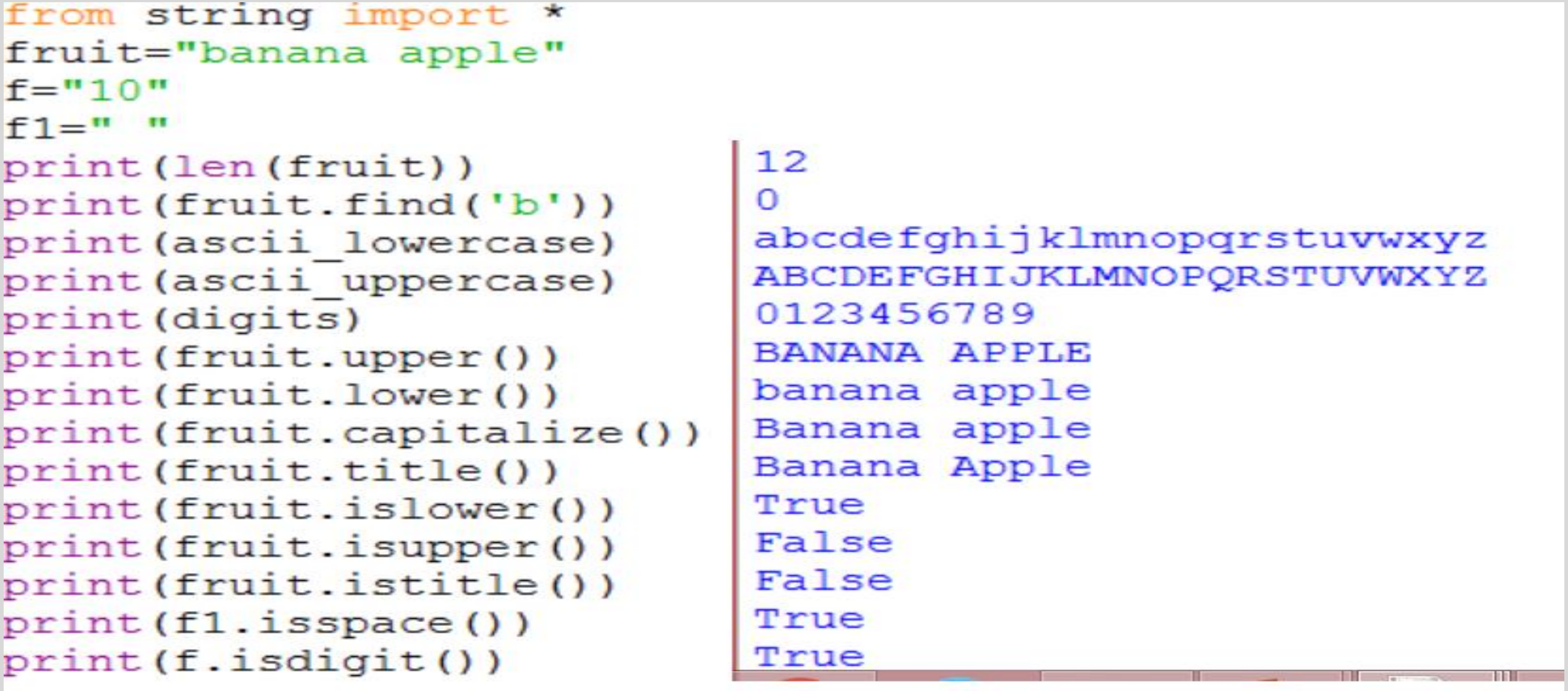
Try out

string.find(“banana”,”na”) **#2**

string.find(“banana”,”na”,3) **#4, (starts from index 3)**

string.find(“bob”,”b”,1,2) #-1, (checks between 1 to 2 excluding 2 index)

String Operations



```
from string import *
fruit="banana apple"
f="10"
f1=" "
print(len(fruit))
print(fruit.find('b'))
print(ascii_lowercase)
print(ascii_uppercase)
print(digits)
print(fruit.upper())
print(fruit.lower())
print(fruit.capitalize())
print(fruit.title())
print(fruit.islower())
print(fruit.isupper())
print(fruit.istitle())
print(f1.isspace())
print(f.isdigit())
```

```
12
0
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789
BANANA APPLE
banana apple
Banana apple
Banana Apple
True
False
False
True
True
```

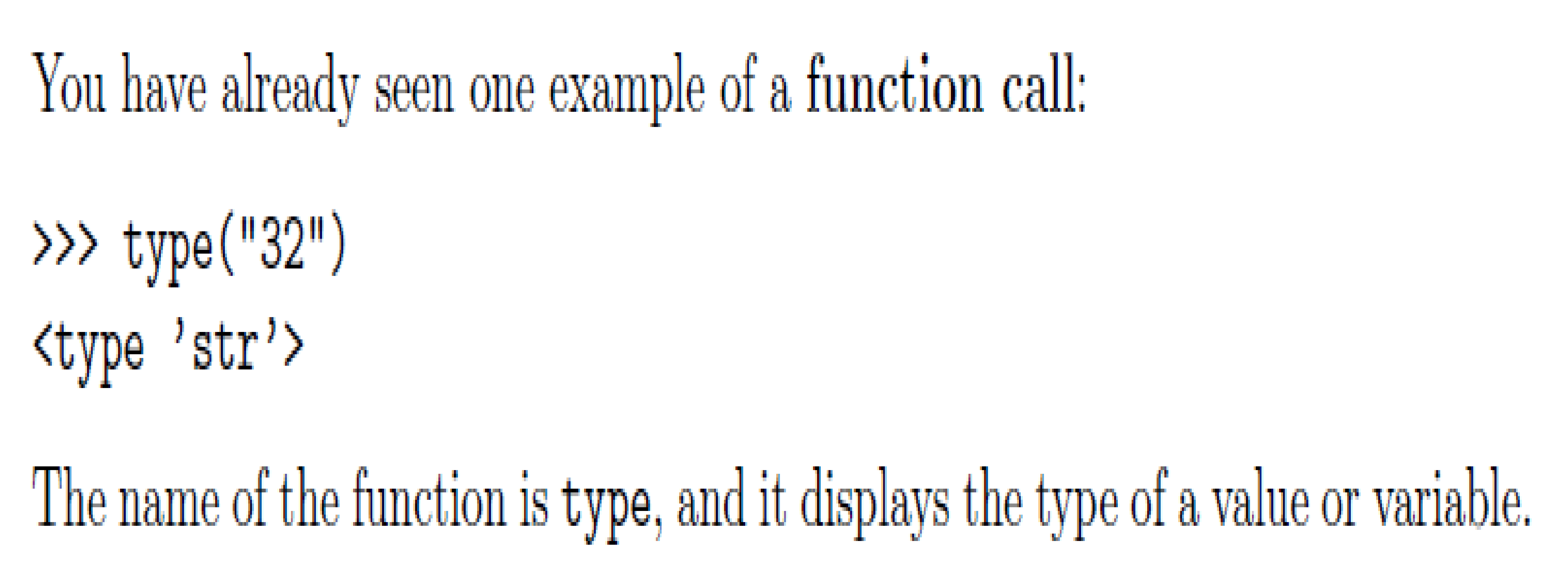
Functions and Recursion

- **Function:** A named sequence of statements that performs some useful operation. Functions may or may not take arguments and may or may not produce a result.

- **Function definition:** A statement that creates a new function, specifying its name, parameters, and the statements it executes.

- **Function call:** A statement that call a function definition to perform a specific task.

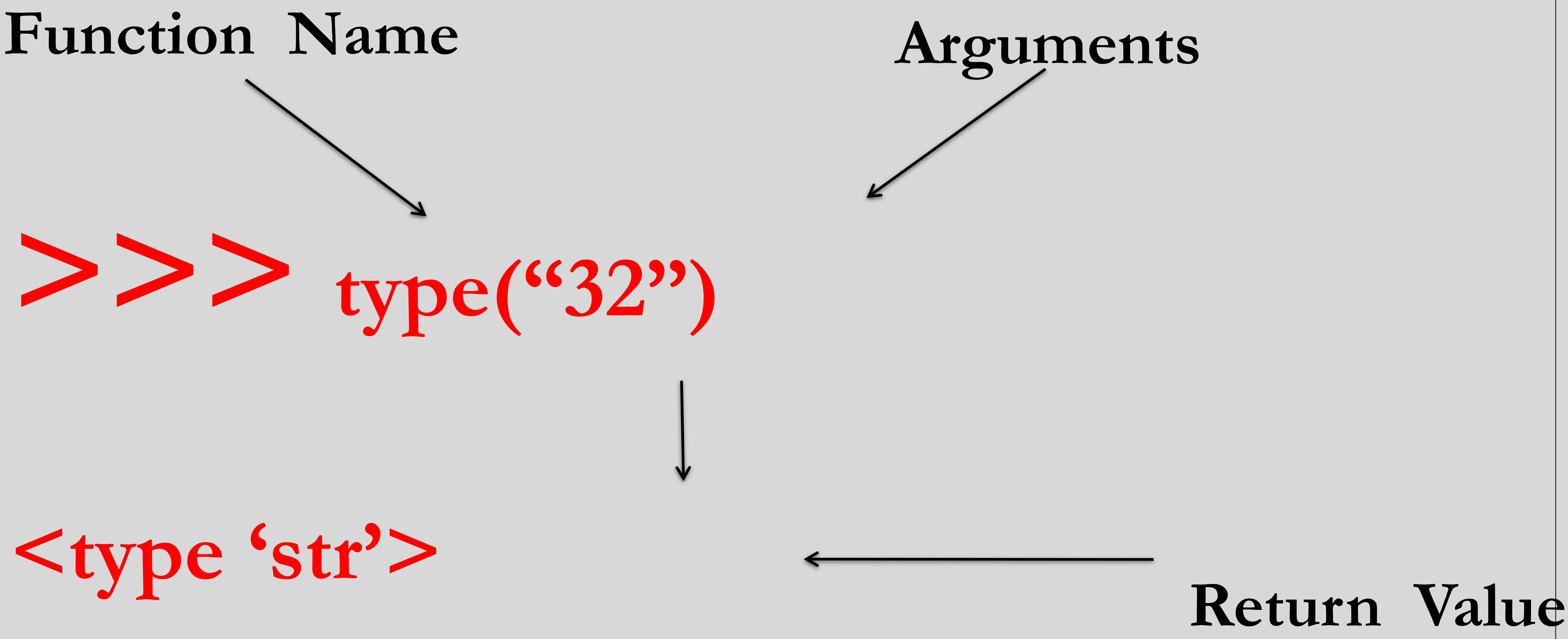
Function calls



You have already seen one example of a function call:

```
>>> type("32")
<type 'str'>
```

The name of the function is `type`, and it displays the type of a value or variable.



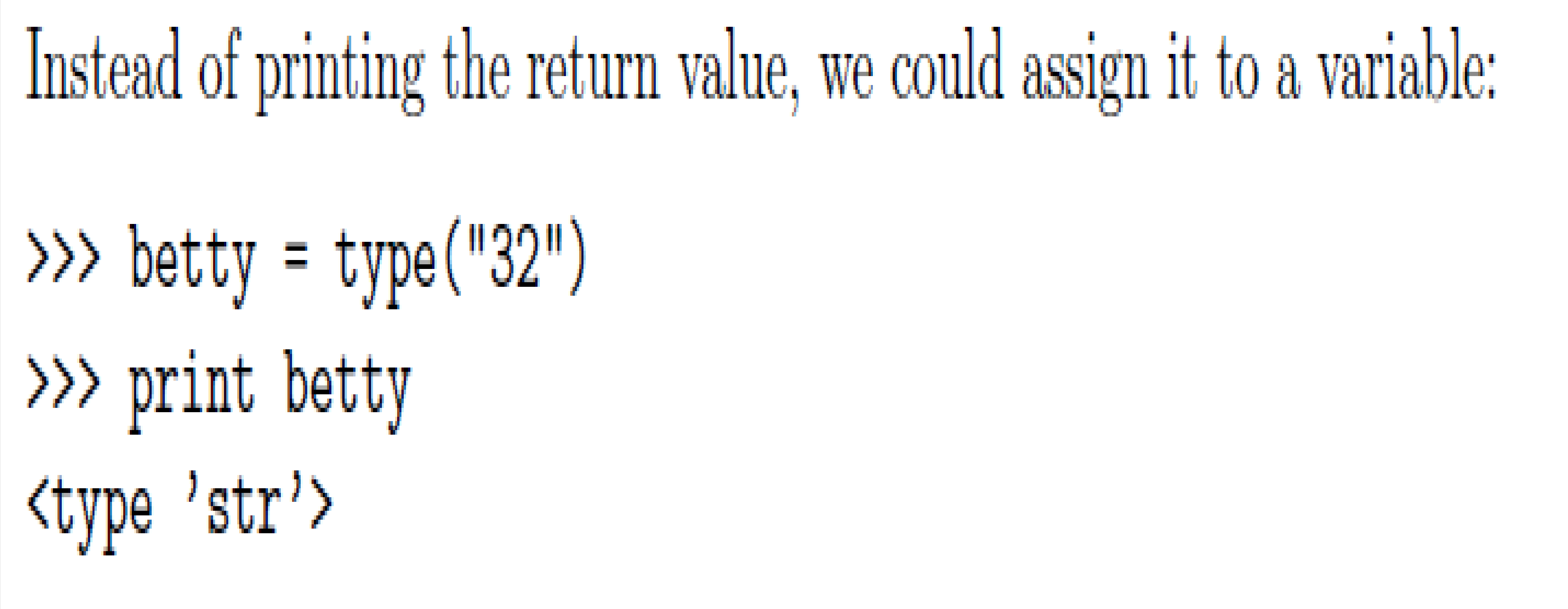
Function Name

Arguments

```
>>> type(“32”)

<type ‘str’>
```

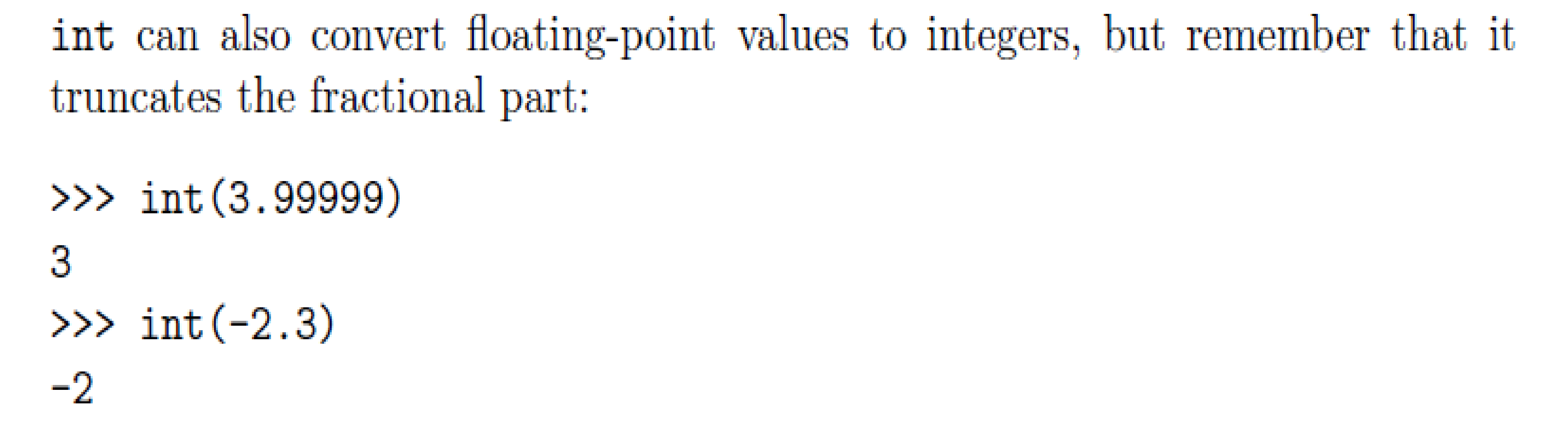
Return Value



Instead of printing the return value, we could assign it to a variable:

```
>>> betty = type("32")
>>> print betty
<type 'str'>
```

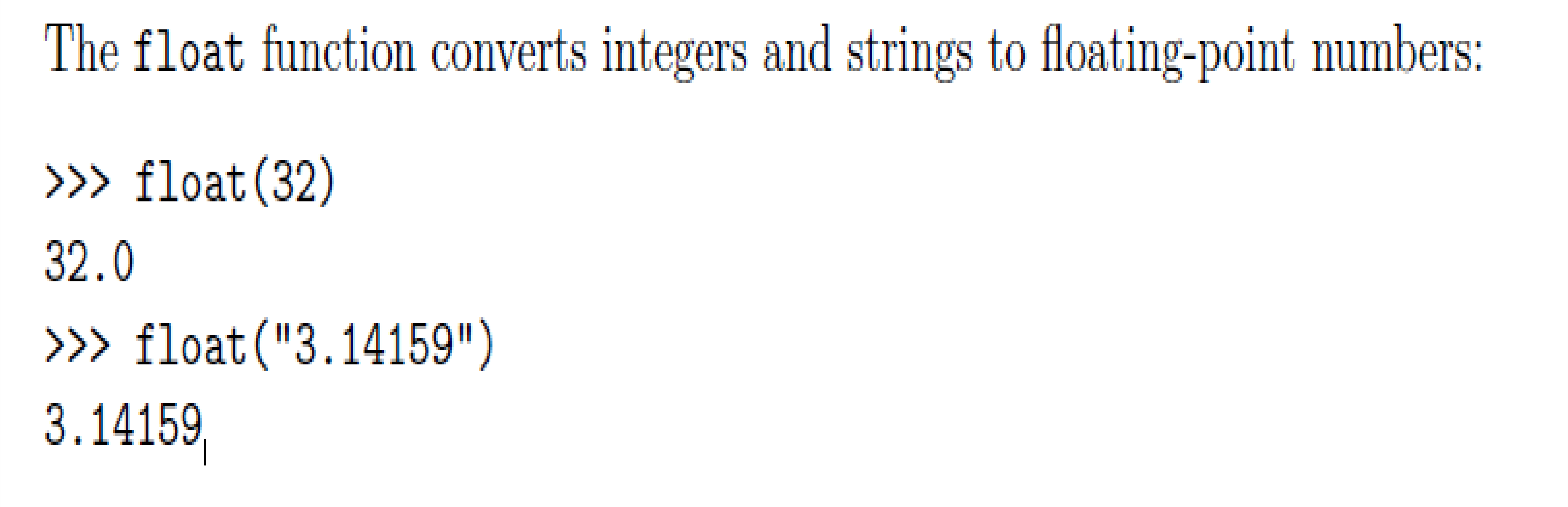
Float to integer conversion



`int` can also convert floating-point values to integers, but remember that it truncates the fractional part:

```
>>> int(3.99999)
3
>>> int(-2.3)
-2
```

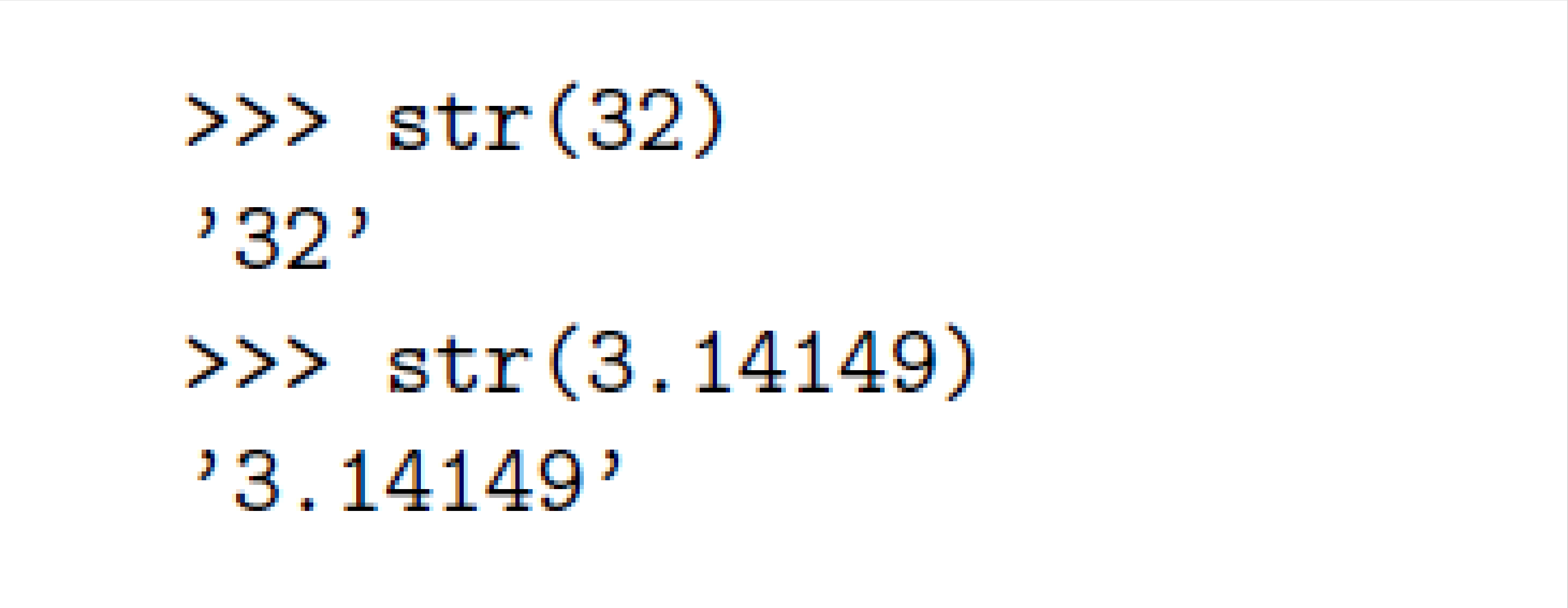
Integer and String Conversion to Float



The `float` function converts integers and strings to floating-point numbers:

```
>>> float(32)
32.0
>>> float("3.14159")
3.14159
```

Integer and Float Conversion to String



```
>>> str(32)
'32'
>>> str(3.14149)
'3.14149'
```

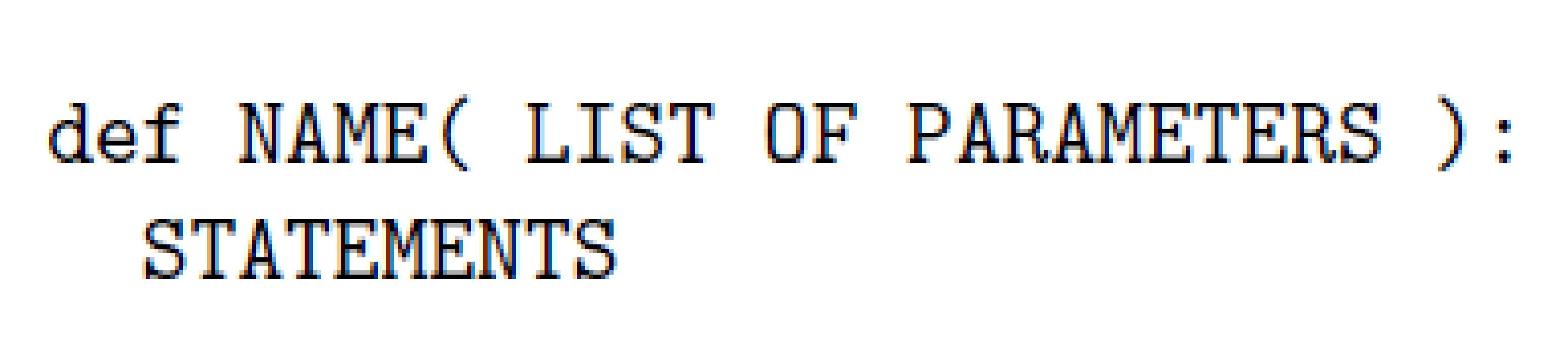
Mathematical Functions

- abs(x)
- Ceil(x)
- cmp (x, y)
- exp (x)
- Floor(x)
- Log(x)
- pow (x,y)
- sqrt(x,y)
- Max (x1,x2….xn)
- Min (x1,x2…xn)

Adding new or user defined functions

- A function is a named sequence of statements that performs a desired operation. This operation is specified in a **function definition**.

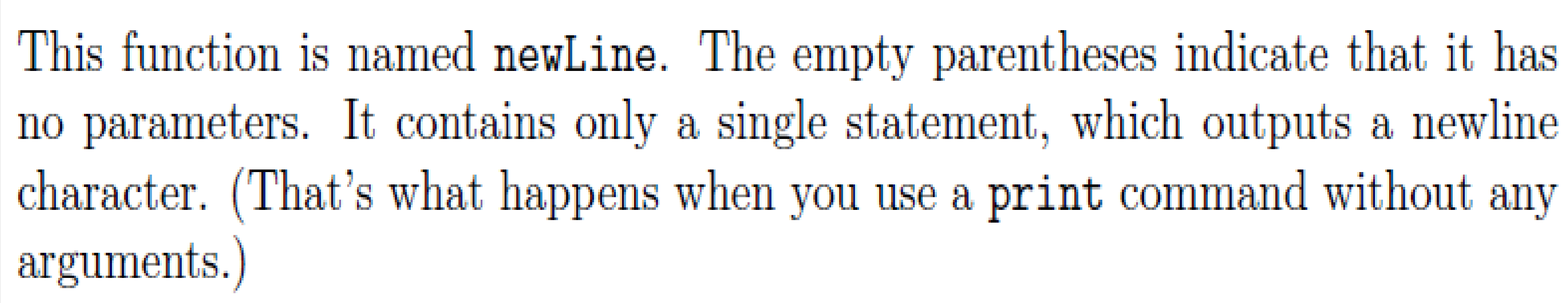
- The syntax for a function definition is:



```
def NAME( LIST OF PARAMETERS ):
  STATEMENTS
```

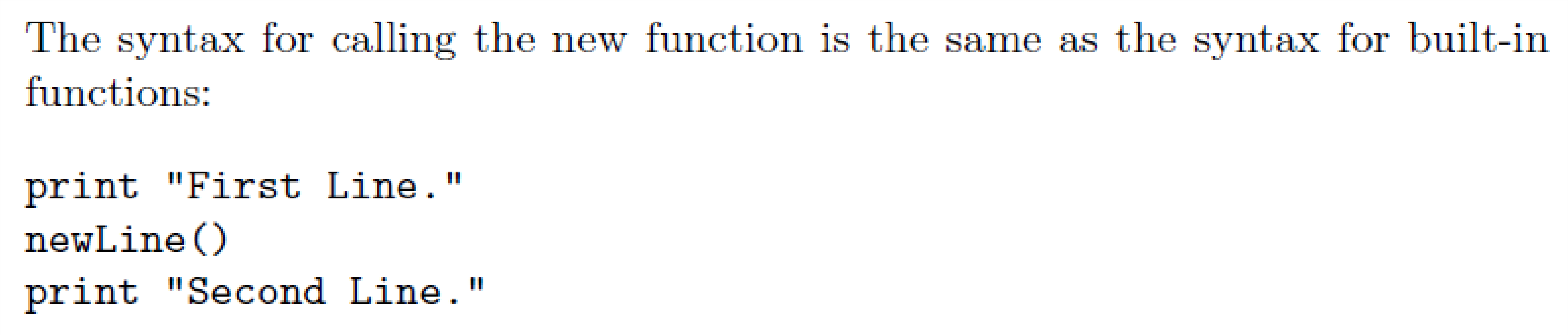
- Example,

```
def newLine():
        Print
```



This function is named newLine. The empty parentheses indicate that it has no parameters. It contains only a single statement, which outputs a newline character. (That's what happens when you use a print command without any arguments.)

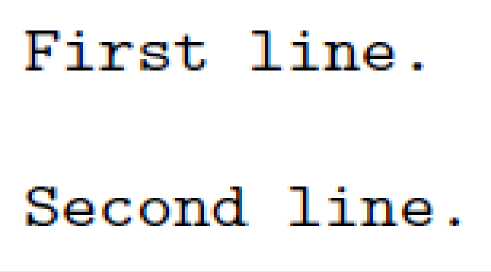
Calling user defined functions



The syntax for calling the new function is the same as the syntax for built-in functions:

```
print "First Line."
newLine()
print "Second Line."
```

- **Output of the program is:**



```
First line.

Second line.
```

Why we need to create functions

- Creating a new function gives you an opportunity to name a group of statements. Functions can simplify a program by hiding a complex computation behind a single command and by using English words in place of arcane code.

- Creating a new function can make a program smaller by eliminating repetitive code.

Few points to remember

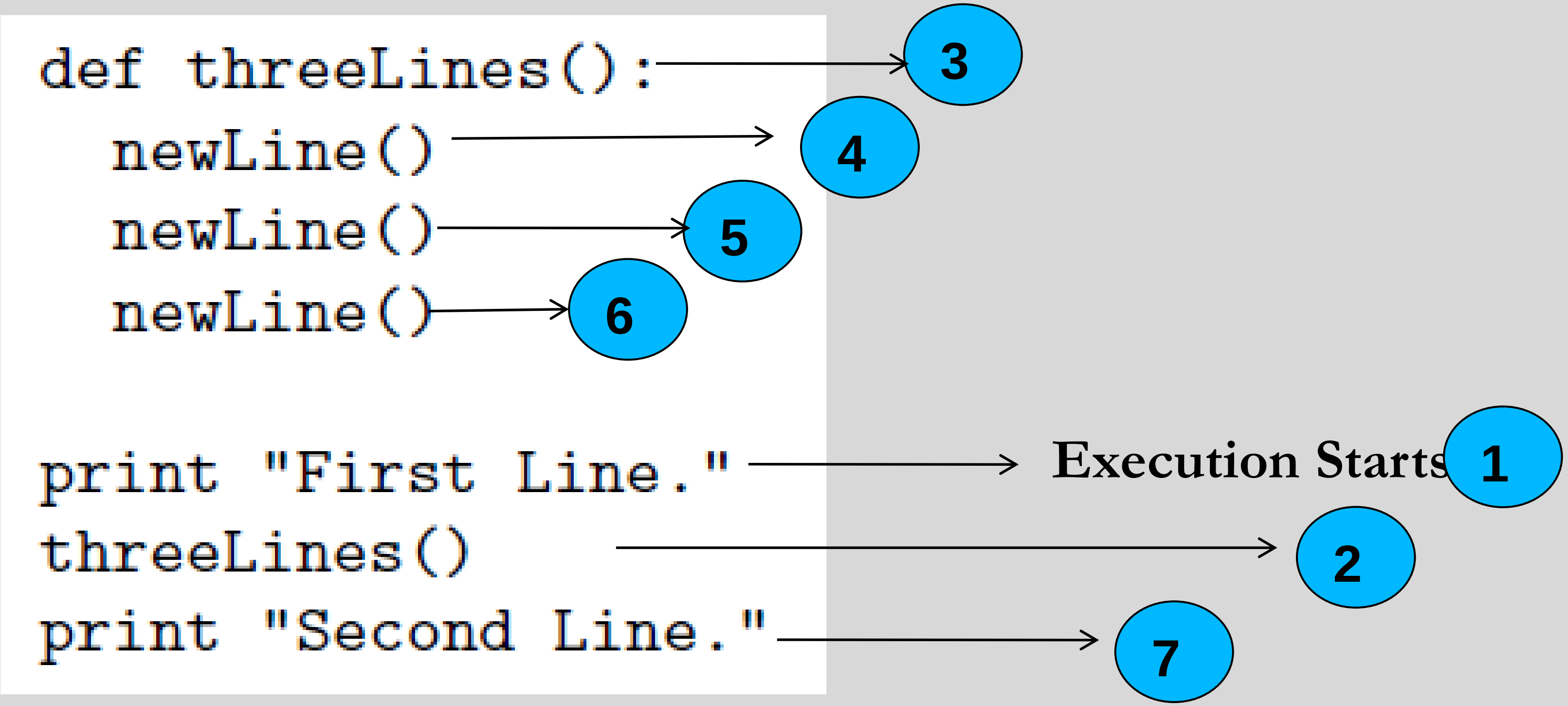
- Only the function definition generates no output.

- The statements inside the function do not get executed until the function is called.

- You must create a function before you can execute it. In other words, the function definition must be executed before the first time it is called.

Flow of execution

- In order to ensure that a function is defined before its first use, you have to know the order in which statements are executed, which is called the flow of execution.

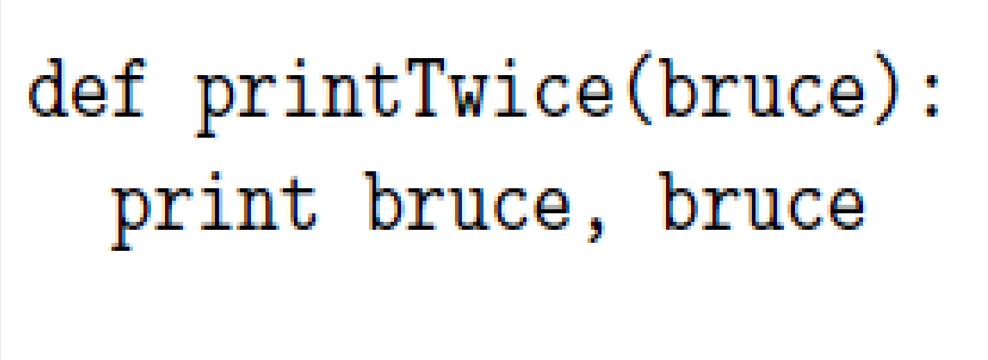


```
def threeLines():
  newLine()
  newLine()
  newLine()

print "First Line."
threeLines()
print "Second Line."
```

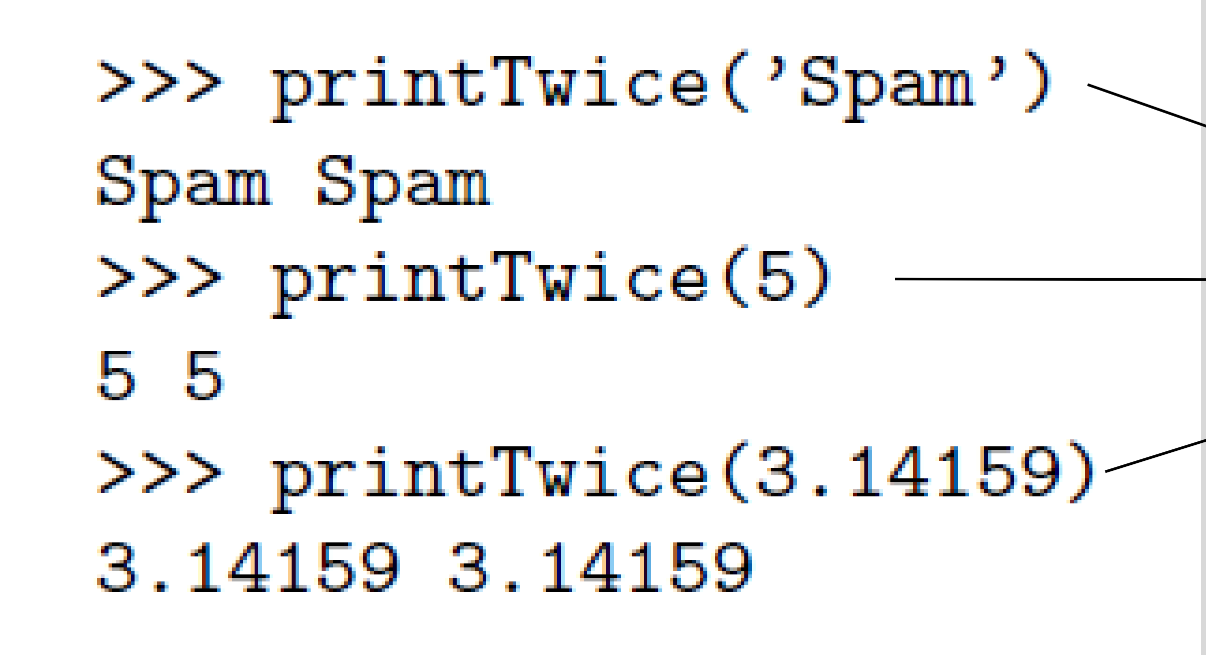
Parameters and arguments

- **Arguments** are the values that control how the function does its job.
- For example, if you want to find the sine of a number, you must indicate what the number is. Thus, sin takes a numeric value as an argument.
- Some functions take more than one argument. For example, pow takes two arguments, the base and the exponent. Inside the function, the values that are passed get assigned to variables called **parameters**.



```
def printTwice(bruce):
  print bruce, bruce
```

Function Definition
with arguments



```
>>> printTwice('Spam')
Spam Spam
>>> printTwice(5)
5 5
>>> printTwice(3.14159)
3.14159 3.14159
```

Function Call
with arguments of
type String, Integer
and float respectively

Functions with results

```
 def  sum (x,y):
       return x+y


>>> a=sum(8,9)
>>> print a
17
```

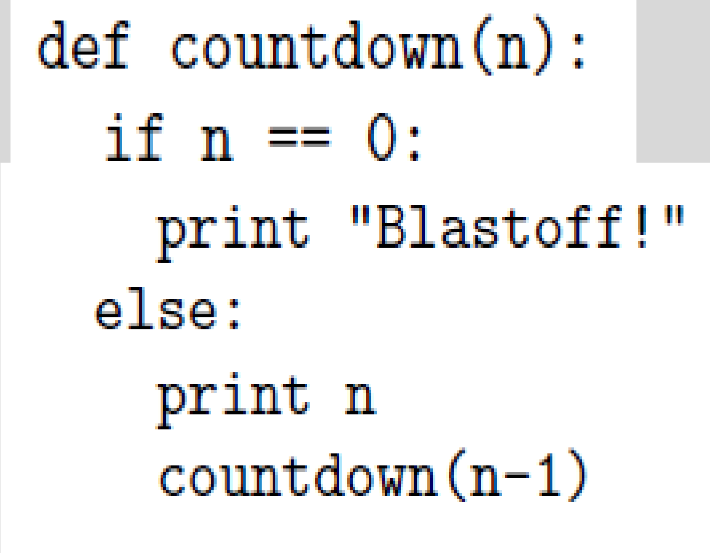
Recursion

- It is legal for one function to call another, and you have seen several examples of that.
- But it is also legal for a function to call itself.

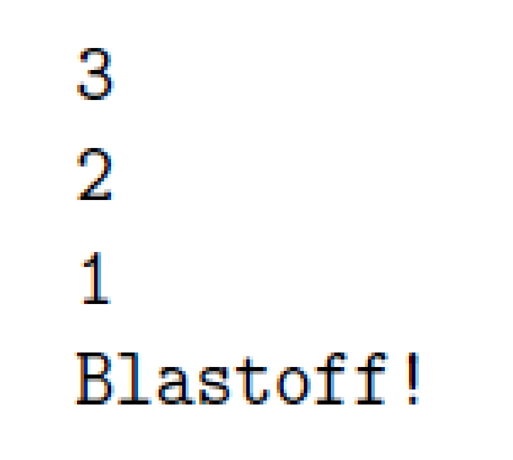
For example,

>>> countdown(3)

OUTPUT of this function will be :



```
def countdown(n):
  if n == 0:
    print "Blastoff!"
  else:
    print n
    countdown(n-1)
```



```
3
2
1
Blastoff!
```

OVERVIEW OF PYTHON LIBRARIES

What is the Python Libraries?

- Library: Tool that you can use to make a specific job.

- A collection of codes or modules of codes that we can use in a program for specific operations. We use libraries so that we don't need to write the code again in our program that is already available.

- The Python Standard Library contains the exact syntax, semantics, and tokens of Python. It contains built-in modules that provide access to basic system functionality like I/O and some other core modules.

Python Libraries

Many popular Python toolboxes/libraries:

- NumPy
- SciPy
- Pandas
- SciKit-Learn

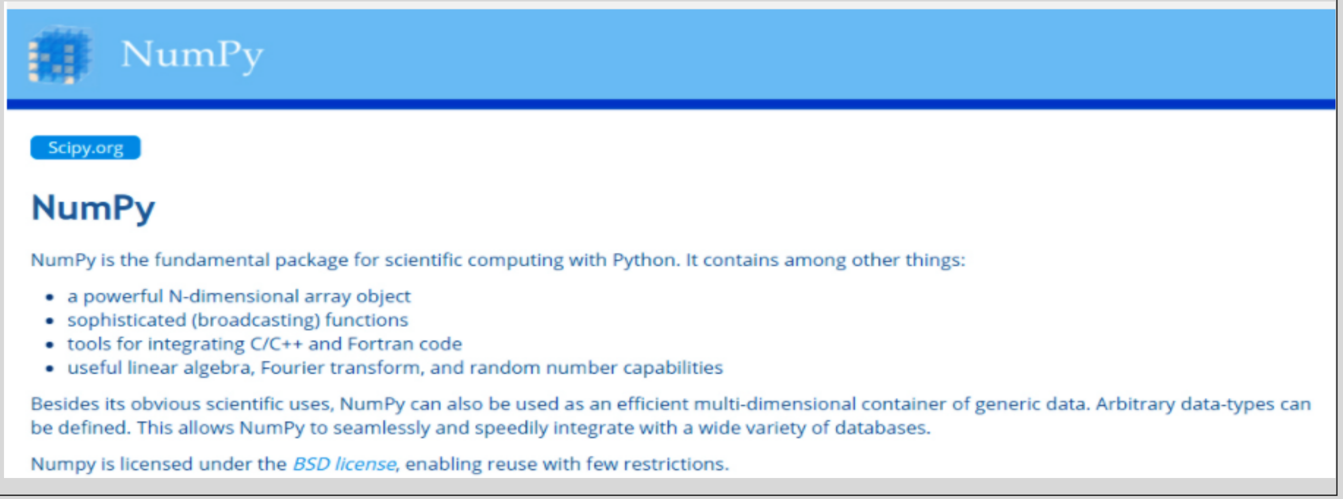
Visualization libraries

- matplotlib
- Seaborn

and many more …

NumPy (Numerical Python)

- The name is an acronym for "Numeric Python" or "Numerical Python".
- NumPy enriches the programming language Python with powerful data structures, implementing multi-dimensional arrays and matrices.
- Perfect tool for scientific computing and performing basic and advanced array operations.
- Introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects
- Many other python libraries are built on NumPy



NumPy

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Numpy is licensed under the *BSD license*, enabling reuse with few restrictions.

Link: http://www.numpy.org/

SciPy

- Includes modules for linear algebra, integration, interpolation, optimization, and statistics.
- SciPy works great for all kinds of scientific programming projects (science, mathematics, and engineering).
- Built on NumPy

Link: https://www.scipy.org/scipylib/

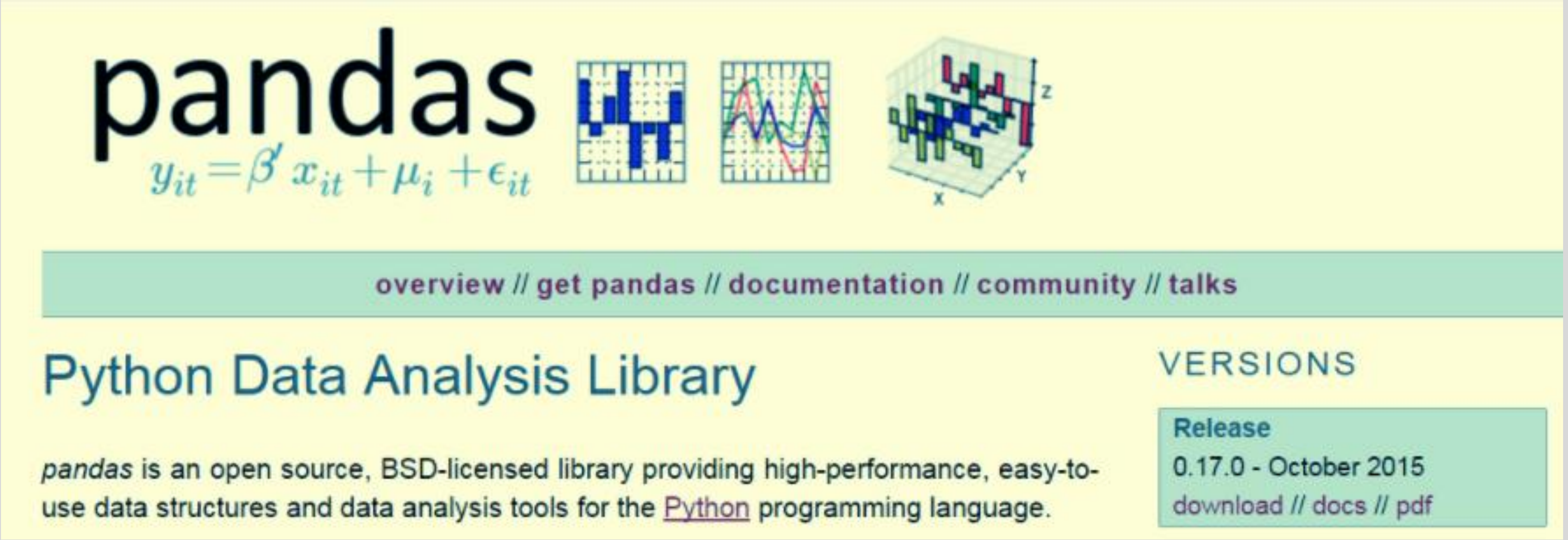
Pandas

- Adds data structures and tools designed to work with table-like data (similar to Series and Data Frames in R)

- Provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.

- It provides special data structures and operations for the manipulation of numerical tables and time series.

- Allows handling missing data.


pandas
$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$
overview // get pandas // documentation // community // talks

Python Data Analysis Library

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

VERSIONS

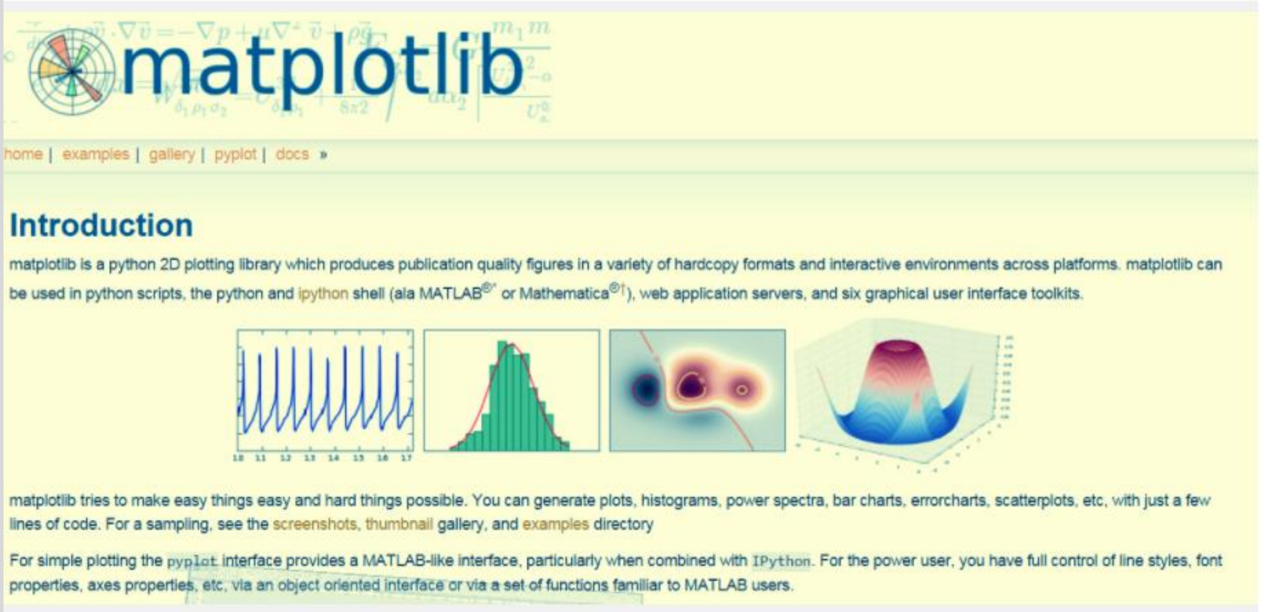
Release
0.17.0 - October 2015
download // docs // pdf

Link: http://pandas.pydata.org/

SciKit-Learn

- Data scientists use it for handling standard machine learning and data mining tasks such as clustering, regression, model selection, dimensionality reduction, and classification.

- Built on NumPy, SciPy and matplotlib.



Introduction

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and ipython shell (ala MATLAB$^{®*}$ or Mathematica$^{®†}$), web application servers, and six graphical user interface toolkits.

matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc, with just a few lines of code. For a sampling, see the screenshots, thumbnail gallery, and examples directory

For simple plotting the `pyplot` interface provides a MATLAB-like interface, particularly when combined with `IPython`. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Link: http://scikit-learn.org/

Matplotlib

- An amazing visualization library in Python for making 2D plots from data in arrays
- Consists of several plots like line, bar, scatter, histogram, pie charts etc.

Link: https://matplotlib.org/

Importing Python Libraries

```
# Importing the libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

THANK YOU!