PYTHON

Nehal Jain
Qcfinance

Introduction

- It was initially designed by Guido van Rossum in 1991 and developed by Python Software Foundation.
- Simple, general purpose, high level, and object-oriented programming language.
- Python is an interpreted scripting language also.
- Python is a dynamic, interpreted (bytecode-compiled) language.
- There are no type declarations of variables, parameters, functions, or methods in source code.
- This makes the code short and flexible, and you lose the compile-time type checking of the source code.
- Python tracks the types of all values at runtime and flags code that does not make sense as it runs.

Why Python?

- ***easy to learn*** yet powerful
- supports ***multiple programming pattern***, including object-oriented, imperative, and functional or procedural programming styles.
- ***multipurpose* programming** language because it can be used with web, enterprise, 3D CAD, etc.
- We don't need to use data types to declare variable because it is ***dynamically typed*** so we can write a=10 to assign an integer value in an integer variable.
- Python makes the development and debugging *fast* because there is no compilation step included in Python development, and **edit-test-debug** cycle is very fast.

Python Applications

1) Web Applications

2) Desktop GUI Applications

3) Software Development

4) Scientific and Numeric

5) Business Applications

6) Console Based Application

7) Audio or Video based Applications

8) 3D CAD Applications

9) Enterprise Applications

10) Applications for Images

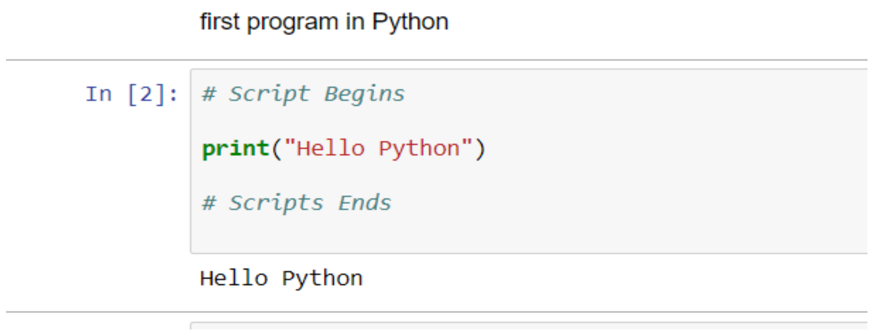
Install Python

- There are two major Python versions- **Python 2 and Python 3**

- link -
- https://www.python.org/downloads/
- https://docs.anaconda.com/anaconda/install/

Writing first program:

- Following is first program in Python



first program in Python

```
In [2]: # Script Begins

        print("Hello Python")

        # Scripts Ends

Hello Python
```

Let us analyze the script line by line.

- ***Line 1 : [*# Script Begins]** In Python comments begin with #. So this statement is for readability of code and ignored by Python interpreter.
- ***Line 2 : [print("Hello Python")]*** In a Python script to print something on the console print() function is used – it simply prints out a line (and also includes a new line).
- ***Line 3 : [*# Script Ends]** This is just another comment like Line 1.

Python Variables

- In Python, we don't need to specify the type of variable because Python is a type infer language and smart enough to get variable type.
- Variable is a name which is used to refer memory location.
- Variable also known as identifier and used to hold value.

Variable Naming

- The first character of the variable must be an alphabet 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).
- Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).
- Identifier name must not be similar to any keyword defined in the language.
- Identifier names are case sensitive

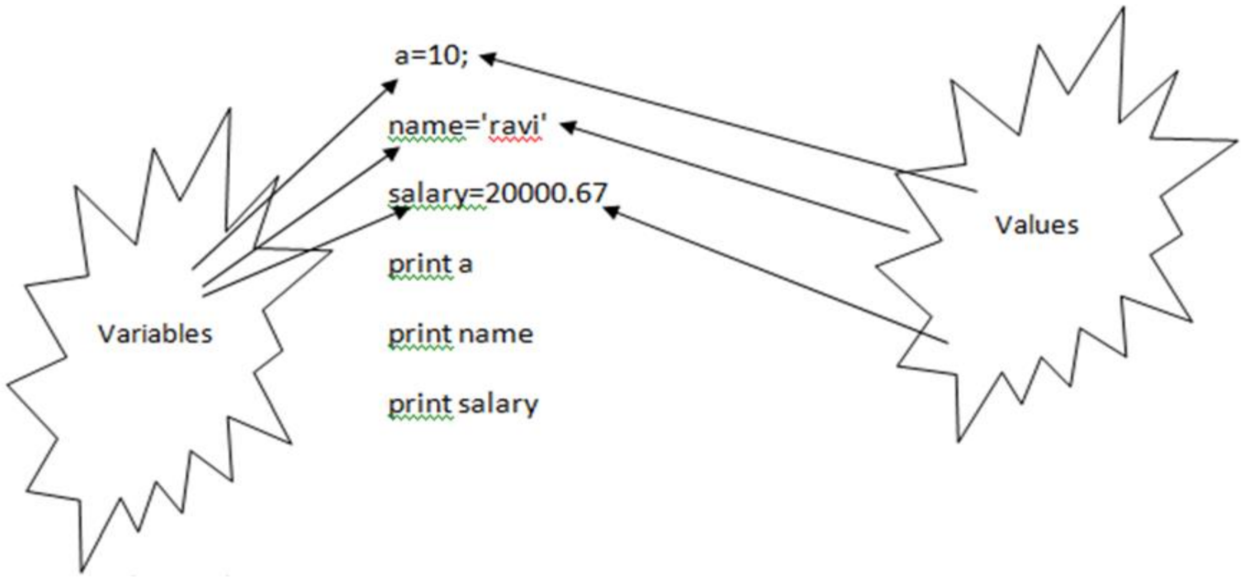
(for example - **my name**, and **MyName** is not the same.)

- Examples of valid identifiers : a123, _n, n_9, etc.
- Examples of invalid identifiers: 1a, n%4, n 9, etc.

Declaring Variable and Assigning Values

- It allows us to create variable at required time.
- When we assign any value to the variable that variable is declared automatically.
- The equal (=) operator is used to assign value to a variable

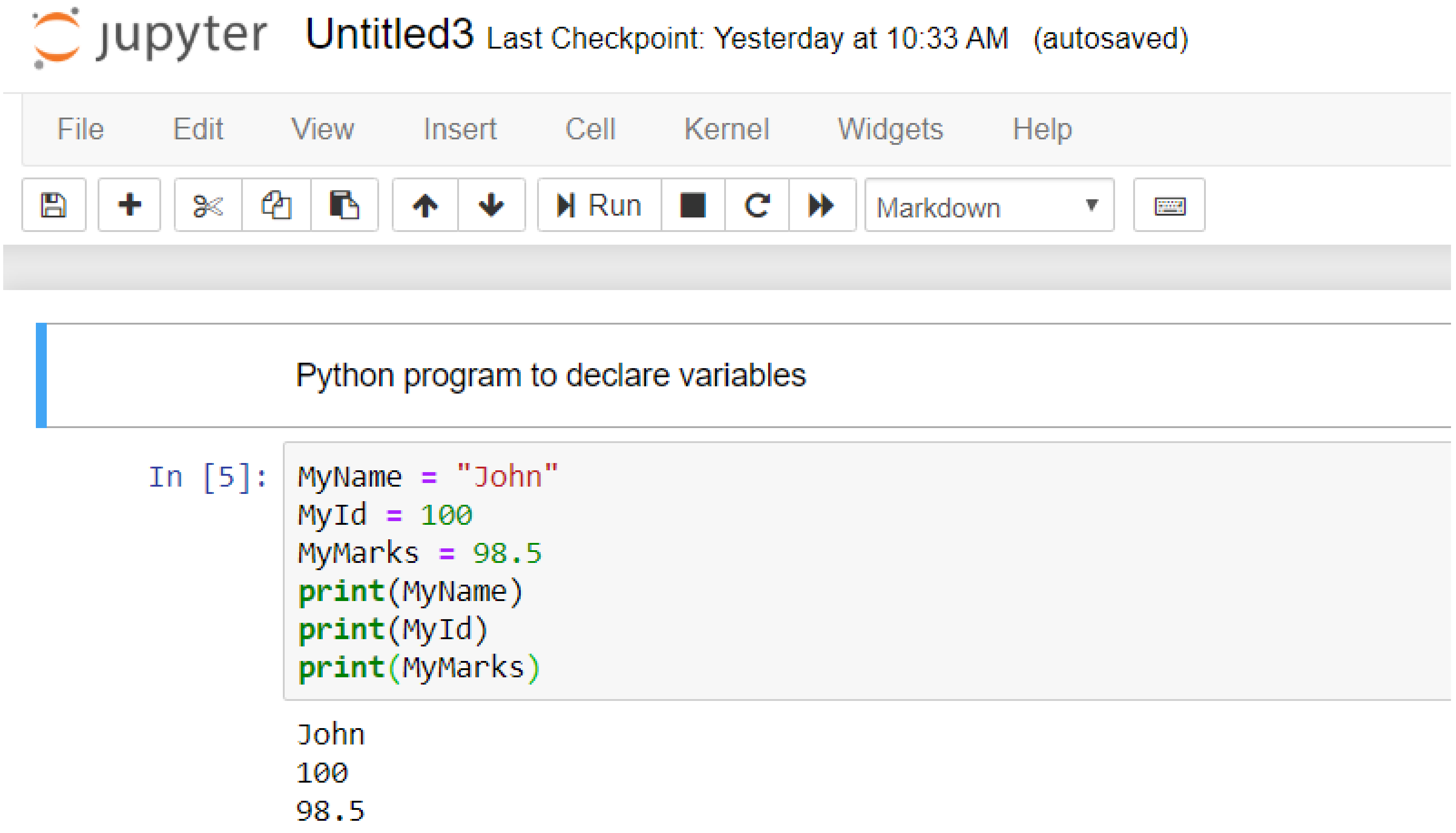
Example-



```
a=10;
name='ravi'
salary=20000.67
print a
print name
print salary
```

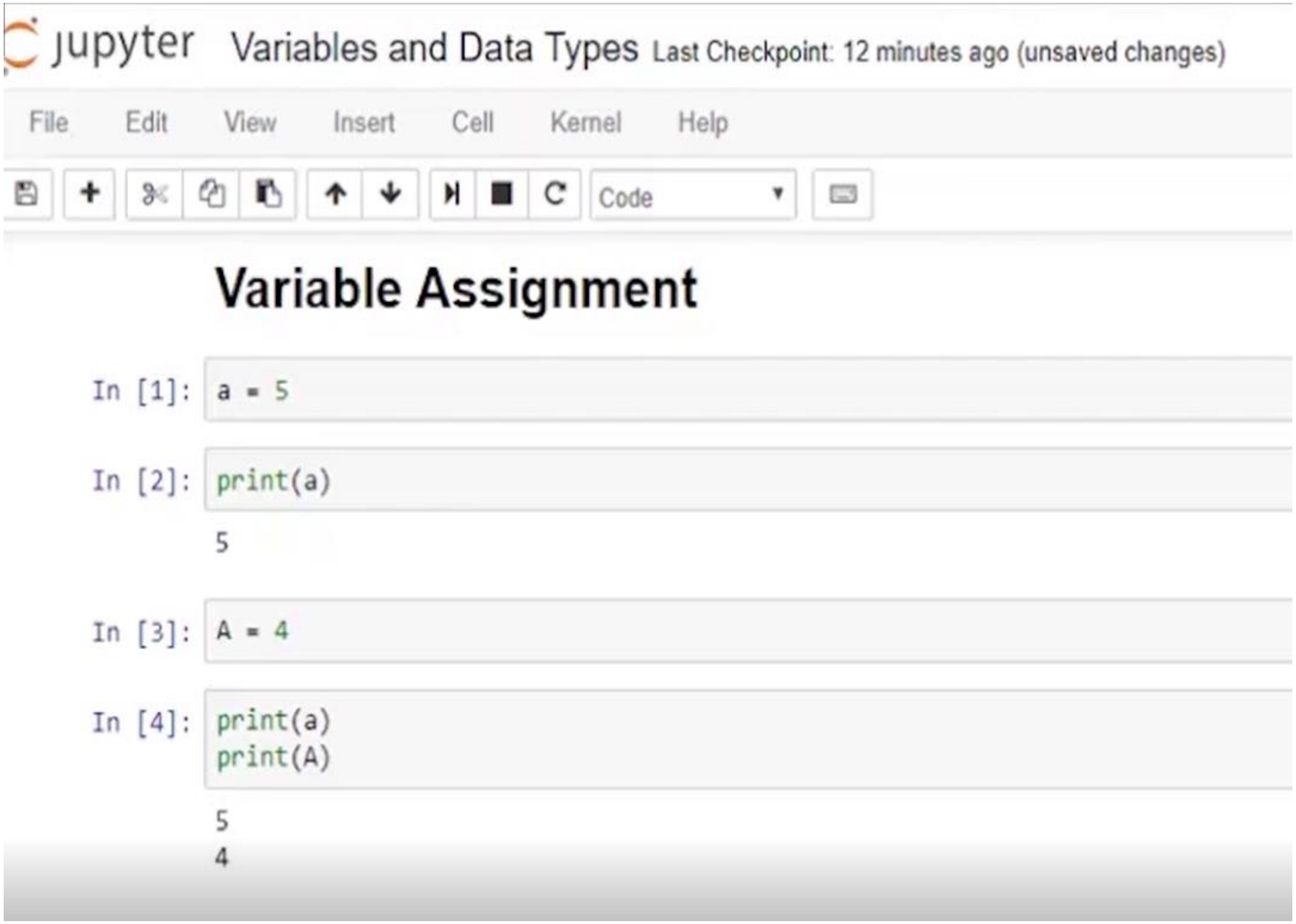
Output:

```
10
ravi
20000.67
```


jupyter
Untitled3 Last Checkpoint: Yesterday at 10:33 AM (autosaved)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Markdown

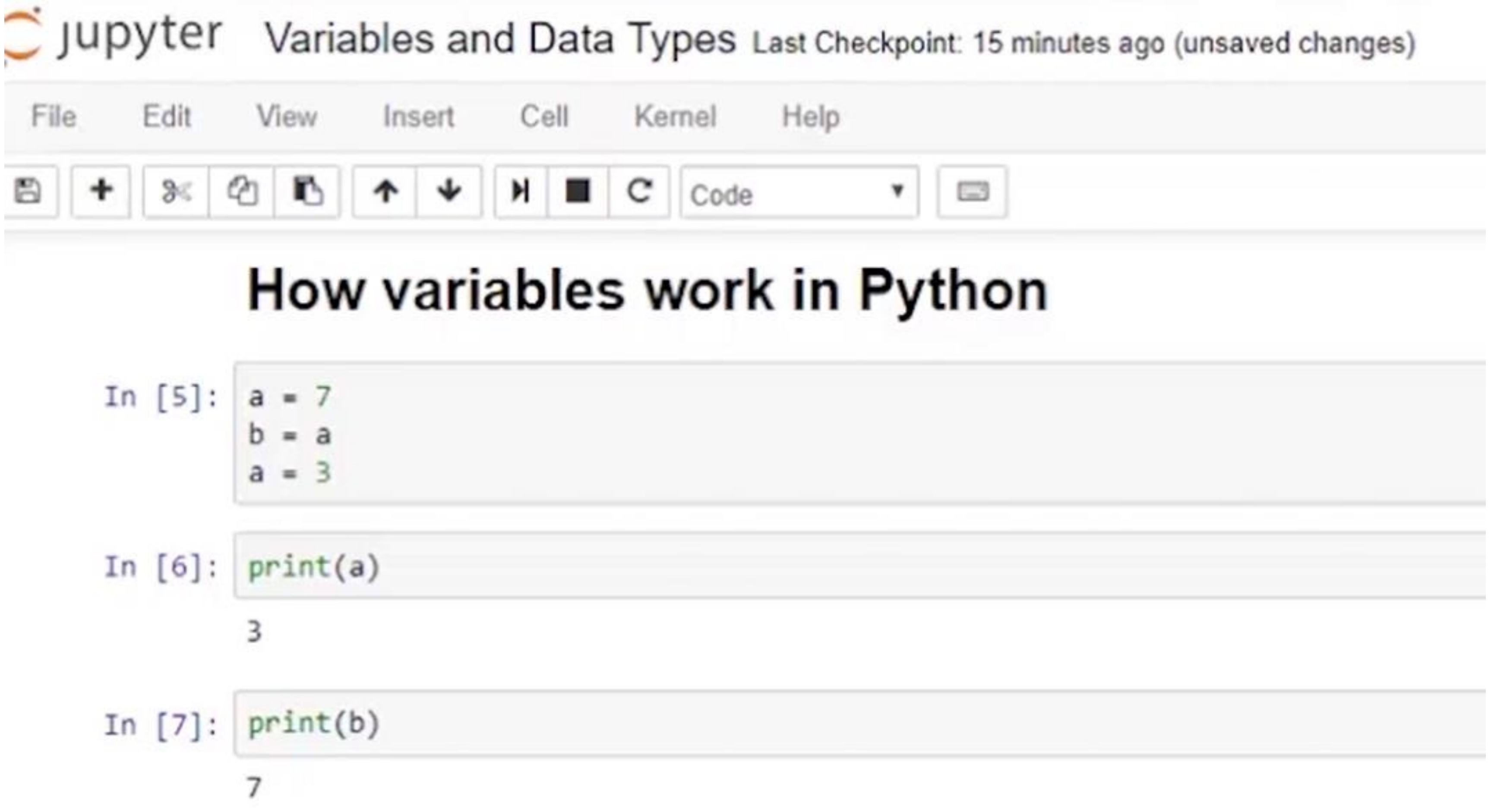
Python program to declare variables

```
In [5]: MyName = "John"
        MyId = 100
        MyMarks = 98.5
        print(MyName)
        print(MyId)
        print(MyMarks)

        John
        100
        98.5
```


jupyter
Variables and Data Types Last Checkpoint: 12 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Help
Code

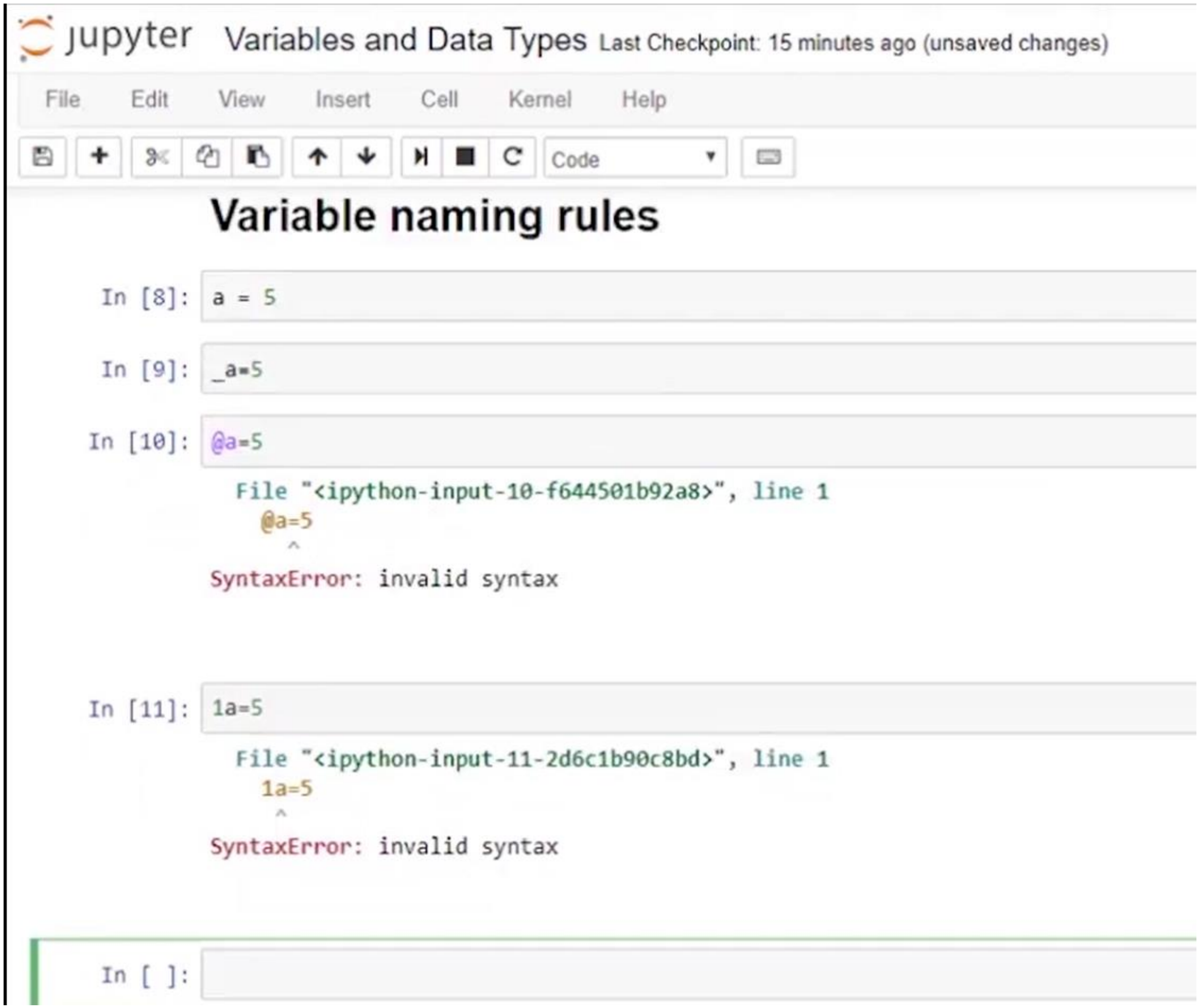
Variable Assignment

```
In [1]: a = 5

In [2]: print(a)

5

In [3]: A = 4

In [4]: print(a)
        print(A)

5
4
```


jupyter
Variables and Data Types Last Checkpoint: 15 minutes ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Help
Code

How variables work in Python

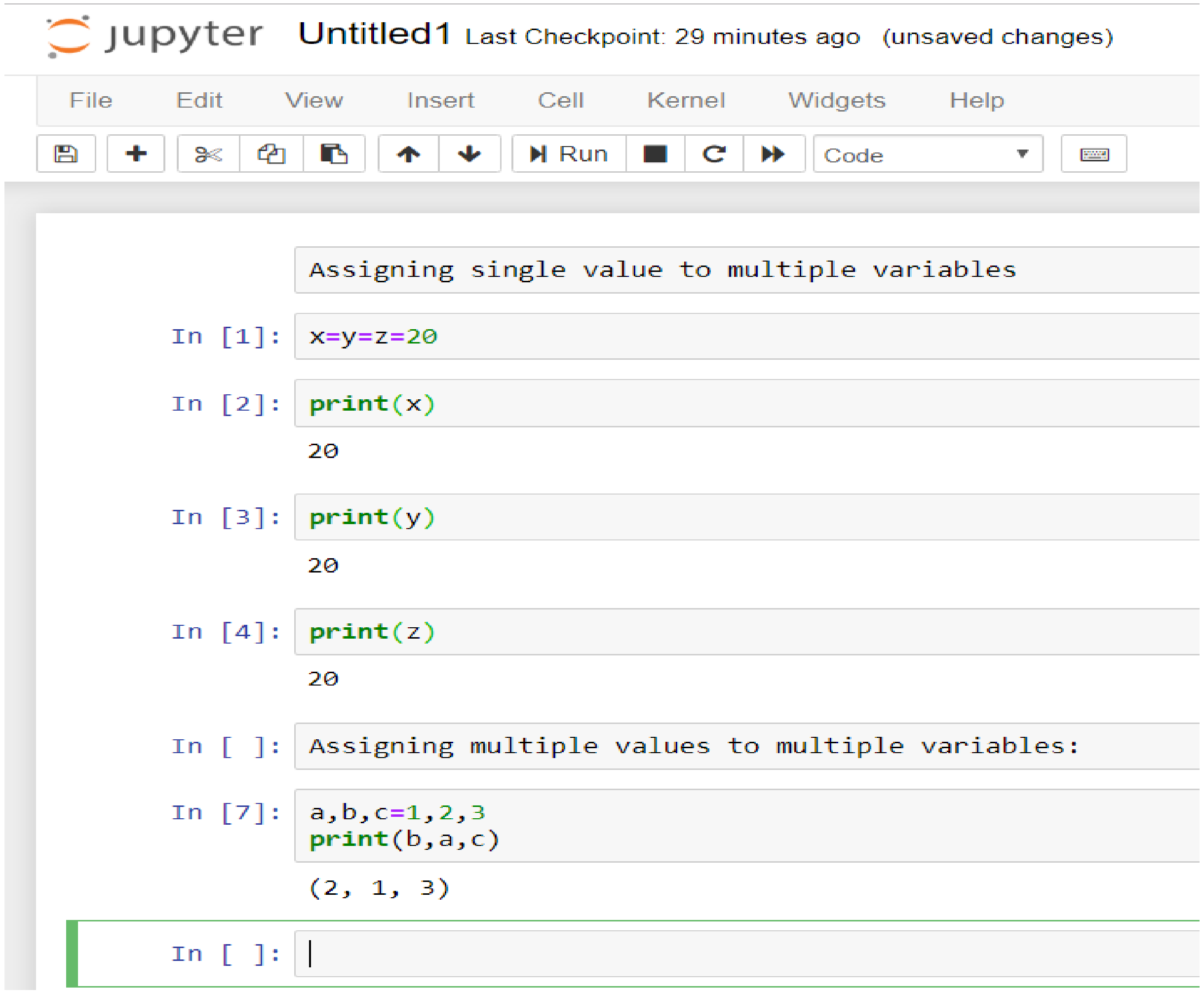
```
In [5]: a = 7
        b = a
        a = 3

In [6]: print(a)

        3

In [7]: print(b)

        7
```


jupyter Variables and Data Types Last Checkpoint: 15 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Help
Code

Variable naming rules

```
In [8]: a = 5

In [9]: _a=5

In [10]: @a=5

  File "<ipython-input-10-f644501b92a8>", line 1
    @a=5
      ^
SyntaxError: invalid syntax

In [11]: 1a=5

  File "<ipython-input-11-2d6c1b90c8bd>", line 1
    1a=5
     ^
SyntaxError: invalid syntax

In [ ]:
```

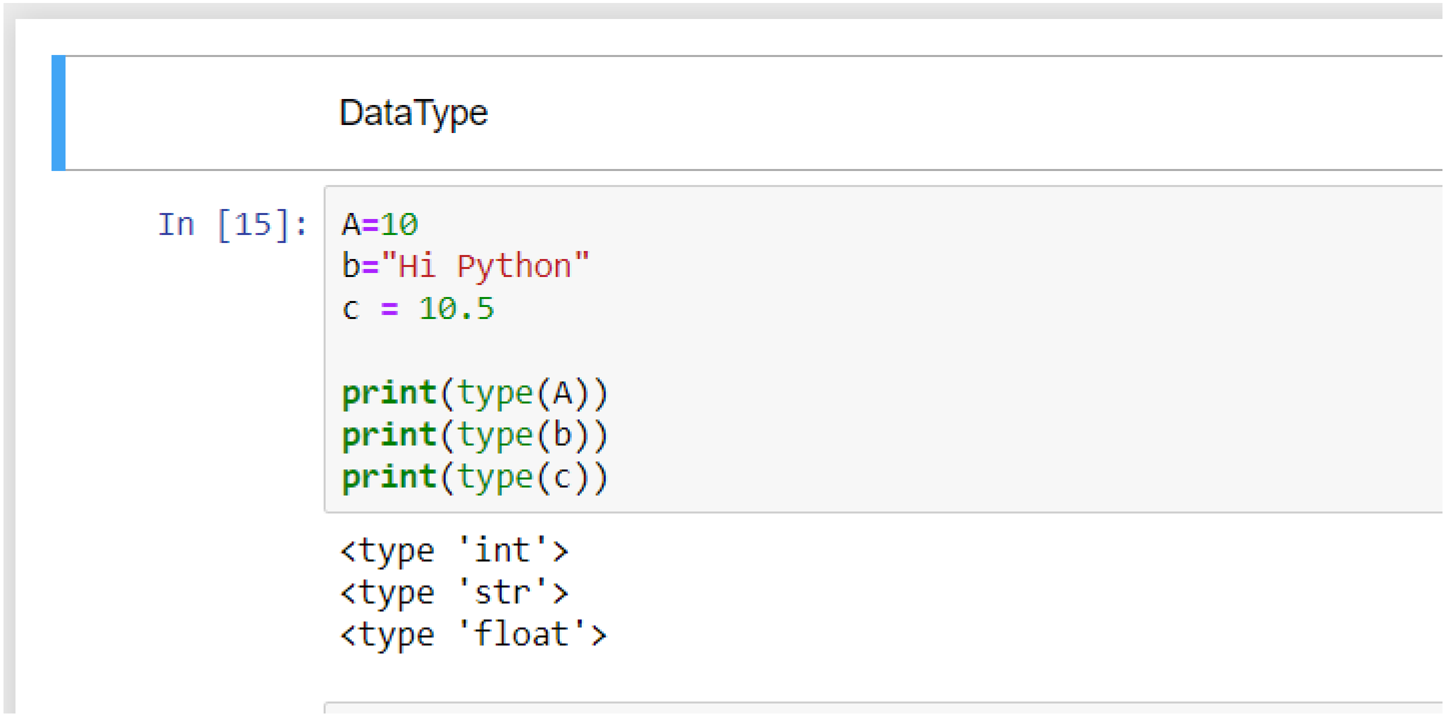
Multiple Assignment


jupyter
Untitled1 Last Checkpoint: 29 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
Assigning single value to multiple variables

In [1]: x=y=z=20

In [2]: print(x)

20

In [3]: print(y)

20

In [4]: print(z)

20

In [ ]: Assigning multiple values to multiple variables:

In [7]: a,b,c=1,2,3
        print(b,a,c)

(2, 1, 3)

In [ ]:
```

Python Data Types

- Python is a dynamically typed language hence we need not define the type of the variable while declaring it.
- The interpreter implicitly binds the value with its type.
- Python enables us to check the type of the variable used in the program.
- Python provides us the **type()** function which returns the type of the variable passed.



DataType

In [15]:

```python
A=10
b="Hi Python"
c = 10.5

print(type(A))
print(type(b))
print(type(c))
```

```
<type 'int'>
<type 'str'>
<type 'float'>
```

Standard data types

- Numbers
- String
- List
- Tuple
- Dictionary

Numbers

- Number stores **numeric values**. Python creates Number objects when a number is assigned to a variable.

  For example;

  a = 3 , b = 5 #a and b are number objects

- Python supports 4 types of numeric data.

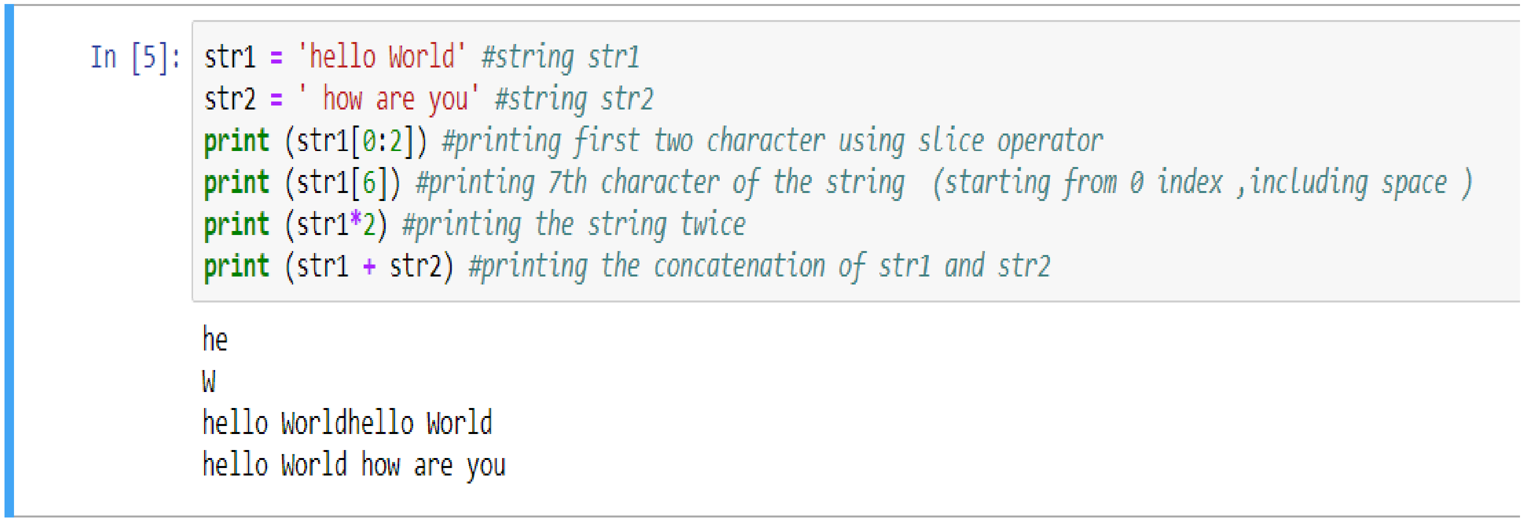
1. **int** (signed integers like 10, 2, 29, etc.)
2. **long** (long integers used for a higher range of values like 908090800L, -0x1929292L, etc.)
3. **float** (float is used to store floating point numbers like 1.9, 9.902, 15.2, etc.)
4. **complex** (complex numbers like 2.14j, 2.0 + 2.3j, etc.)

- A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts respectively).

- Python allows us to use a lower-case L to be used with long integers. However, we must always use an upper-case L to avoid confusion.

String

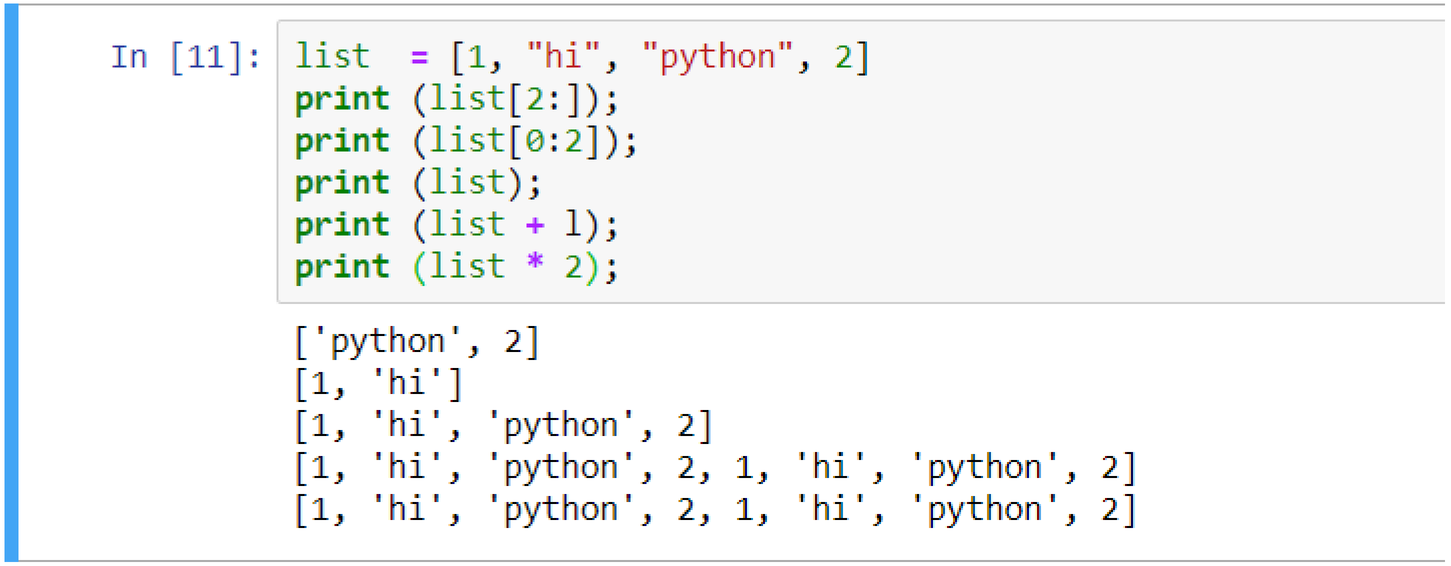
- The string can be defined as the sequence of characters represented in the quotation marks.
- In python, we can use single, double, or triple quotes to define a string.
- In the case of string handling, the operator + is used to concatenate two strings as the operation *"hello"+" python"* returns *"hello python"*.
- The operator * is known as repetition operator as the operation "Python " *2 returns "Python Python ".
-



```
In [5]: str1 = 'hello World' #string str1
        str2 = ' how are you' #string str2
        print (str1[0:2]) #printing first two character using slice operator
        print (str1[6]) #printing 7th character of the string  (starting from 0 index ,including space )
        print (str1*2) #printing the string twice
        print (str1 + str2) #printing the concatenation of str1 and str2

        he
        W
        hello Worldhello World
        hello World how are you
```

List

- List can contain data of different types.
- The items stored in the list are separated with a comma (,) and enclosed within square brackets [].
- We can use slice [:] operators to access the data of the list.
- The concatenation operator (+) and repetition operator (*) works with the list in the same way as they were working with the strings.

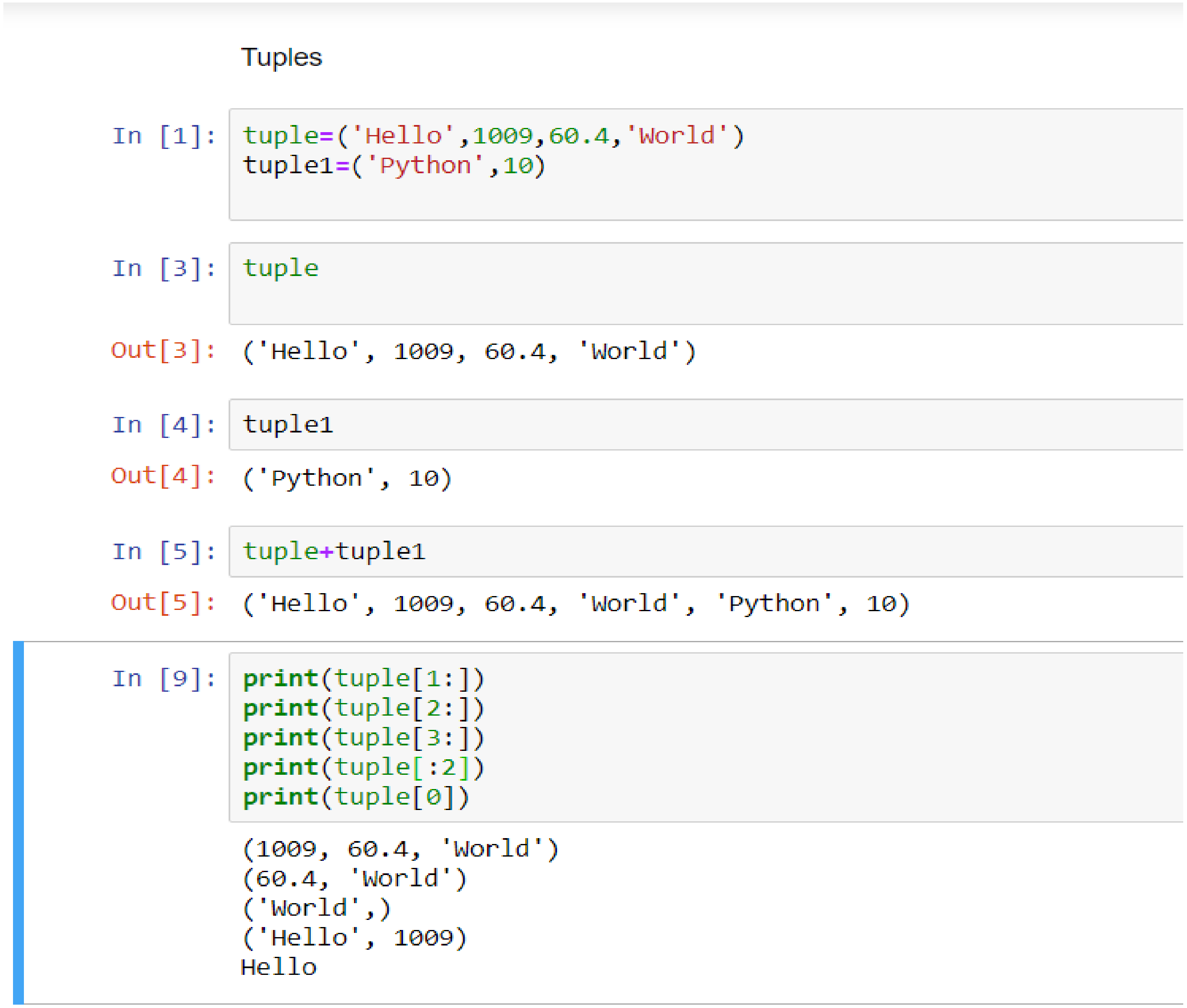


```
In [11]: list  = [1, "hi", "python", 2]
         print (list[2:]);
         print (list[0:2]);
         print (list);
         print (list + l);
         print (list * 2);
```

```
['python', 2]
[1, 'hi']
[1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
[1, 'hi', 'python', 2, 1, 'hi', 'python', 2]
```

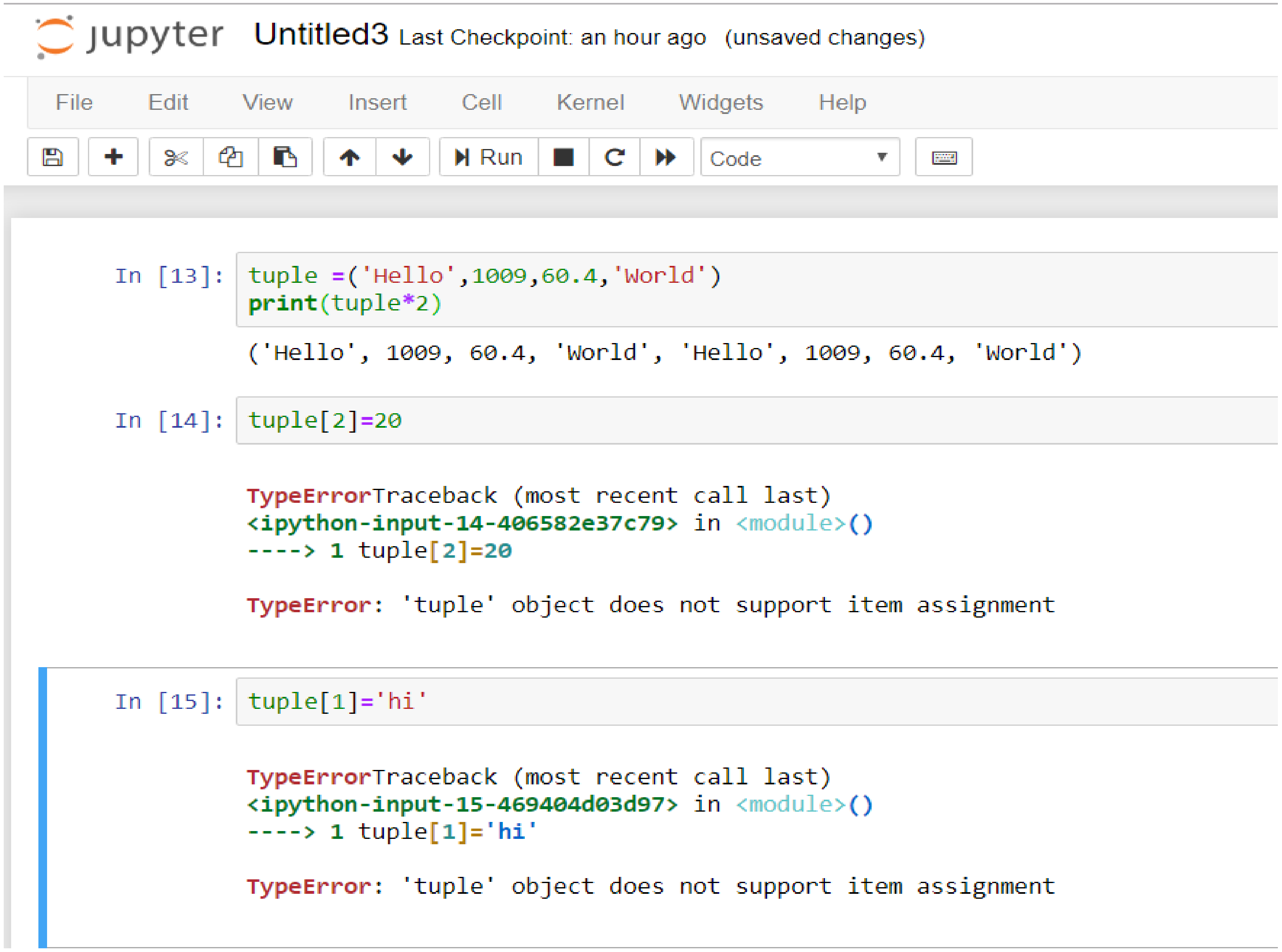
Tuples

- Tuple is another form of collection where different type of data can be stored.
- It is similar to list where data is separated by commas.
- Only the difference is that list uses square bracket [ ] and tuple uses parenthesis().
- Tuples are enclosed in parenthesis and **cannot be changed**.
- A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.



Tuples

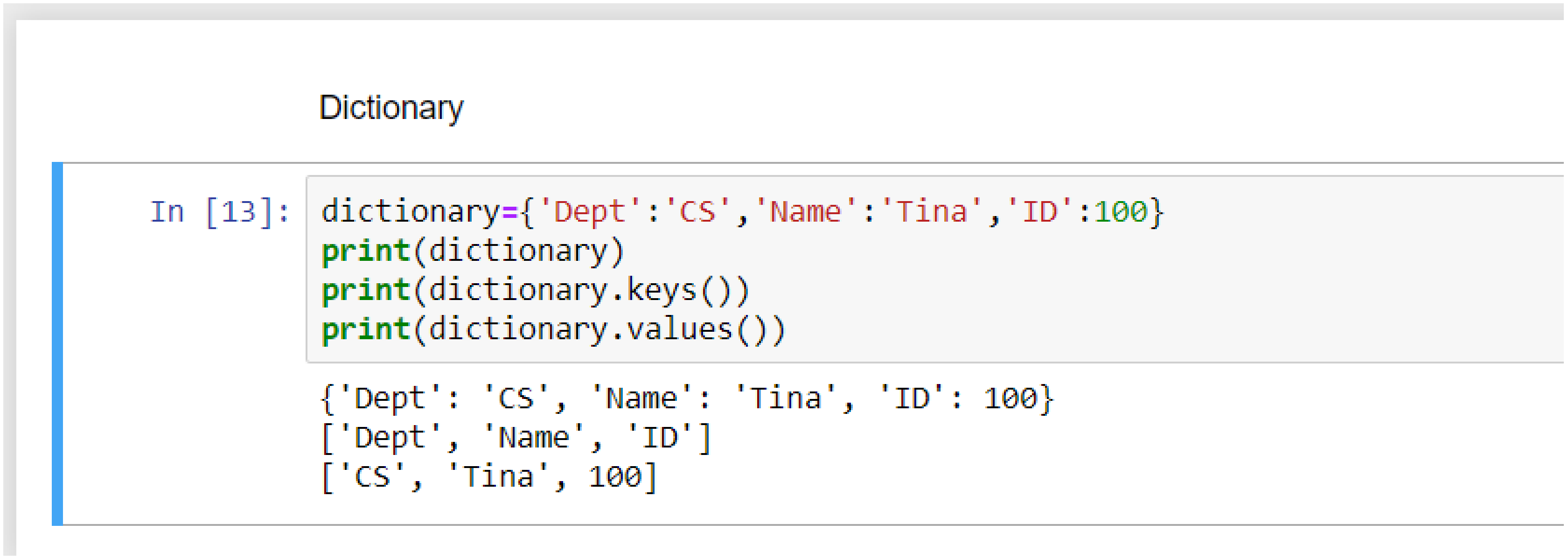
```
In [1]: tuple=('Hello',1009,60.4,'World')
        tuple1=('Python',10)

In [3]: tuple

Out[3]: ('Hello', 1009, 60.4, 'World')

In [4]: tuple1

Out[4]: ('Python', 10)

In [5]: tuple+tuple1

Out[5]: ('Hello', 1009, 60.4, 'World', 'Python', 10)

In [9]: print(tuple[1:])
        print(tuple[2:])
        print(tuple[3:])
        print(tuple[:2])
        print(tuple[0])

        (1009, 60.4, 'World')
        (60.4, 'World')
        ('World',)
        ('Hello', 1009)
        Hello
```


jupyter
Untitled3 Last Checkpoint: an hour ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

```
In [13]: tuple =('Hello',1009,60.4,'World')
         print(tuple*2)

         ('Hello', 1009, 60.4, 'World', 'Hello', 1009, 60.4, 'World')

In [14]: tuple[2]=20

         TypeErrorTraceback (most recent call last)
         <ipython-input-14-406582e37c79> in <module>()
         ----> 1 tuple[2]=20

         TypeError: 'tuple' object does not support item assignment

In [15]: tuple[1]='hi'

         TypeErrorTraceback (most recent call last)
         <ipython-input-15-469404d03d97> in <module>()
         ----> 1 tuple[1]='hi'

         TypeError: 'tuple' object does not support item assignment
```

Dictionary

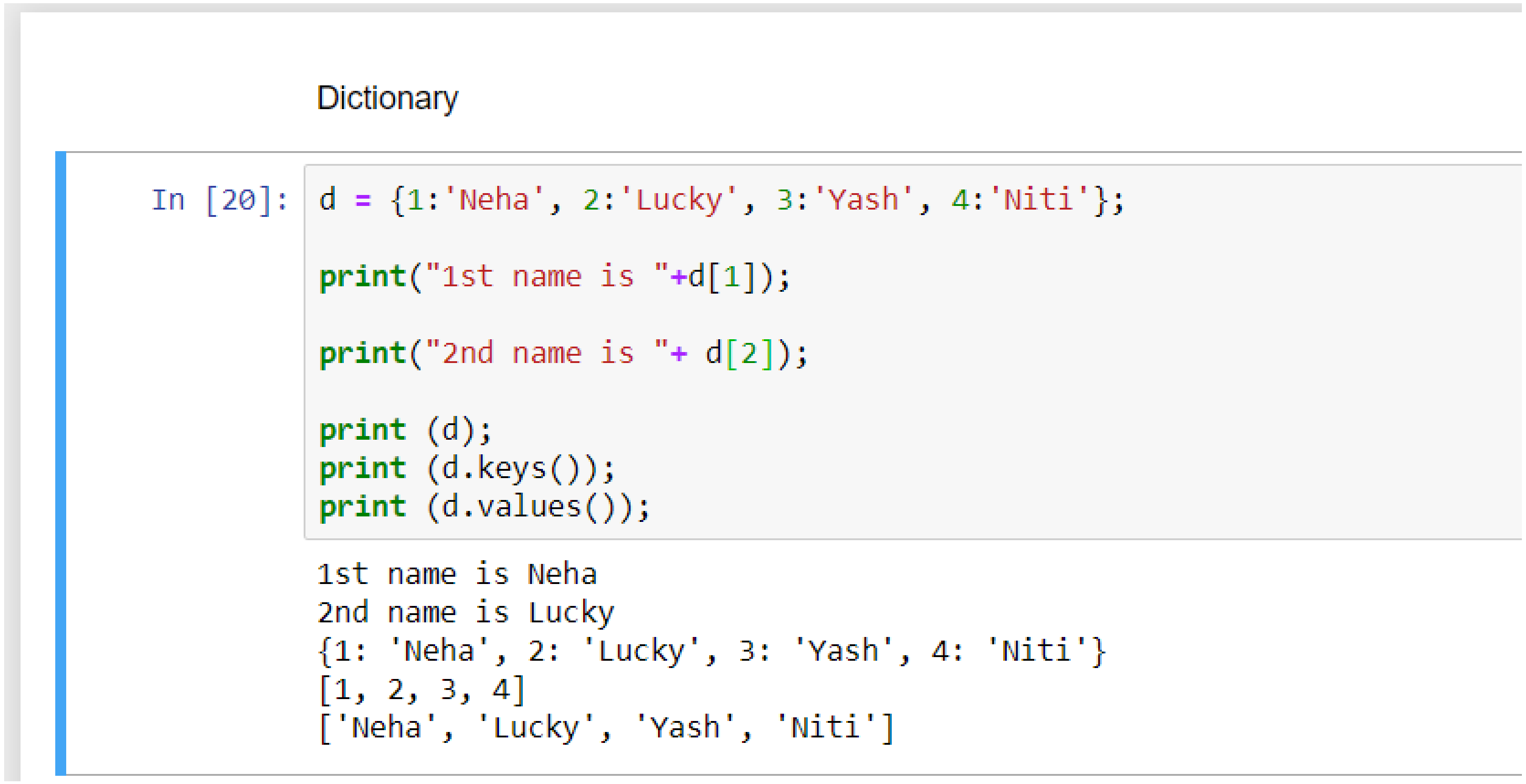
- Dictionary is a collection which works on a key-value pair.
- It works like an associated array where no two keys can be same.
- Dictionaries are enclosed by curly braces ({}) and values can be retrieved by square bracket([]).



Dictionary

In [13]:
```
dictionary={'Dept':'CS','Name':'Tina','ID':100}
print(dictionary)
print(dictionary.keys())
print(dictionary.values())
```

```
{'Dept': 'CS', 'Name': 'Tina', 'ID': 100}
['Dept', 'Name', 'ID']
['CS', 'Tina', 100]
```

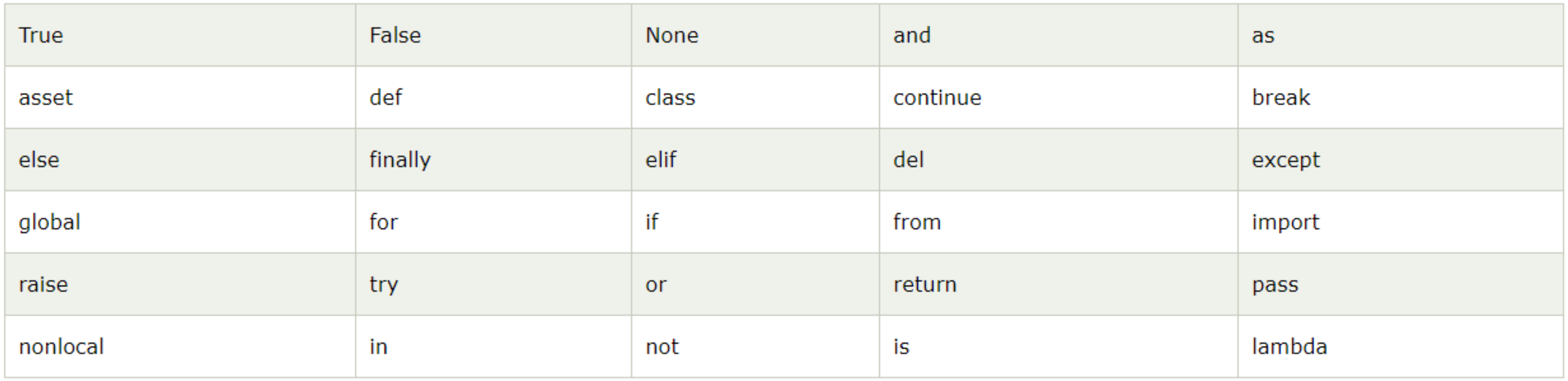

Dictionary

```
In [20]: d = {1:'Neha', 2:'Lucky', 3:'Yash', 4:'Niti'};

         print("1st name is "+d[1]);

         print("2nd name is "+ d[2]);

         print (d);
         print (d.keys());
         print (d.values());
```

```
1st name is Neha
2nd name is Lucky
{1: 'Neha', 2: 'Lucky', 3: 'Yash', 4: 'Niti'}
[1, 2, 3, 4]
['Neha', 'Lucky', 'Yash', 'Niti']
```

Python Keywords

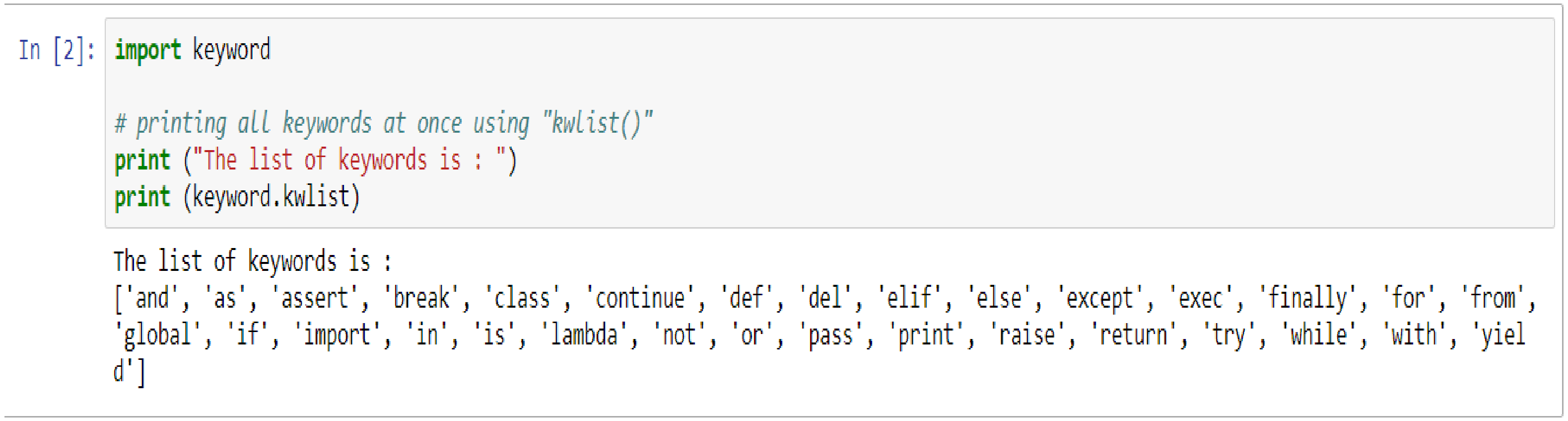
- Python Keywords are special reserved words which convey a special meaning to the compiler/interpreter.
- Each keyword have a special meaning and a specific operation.
- These keywords can't be used as variable.
- Following is the List of Python Keywords.



True	False	None	and	as
asset	def	class	continue	break
else	finally	elif	del	except
global	for	if	from	import
raise	try	or	return	pass
nonlocal	in	not	is	lambda

How to print list of all keywords

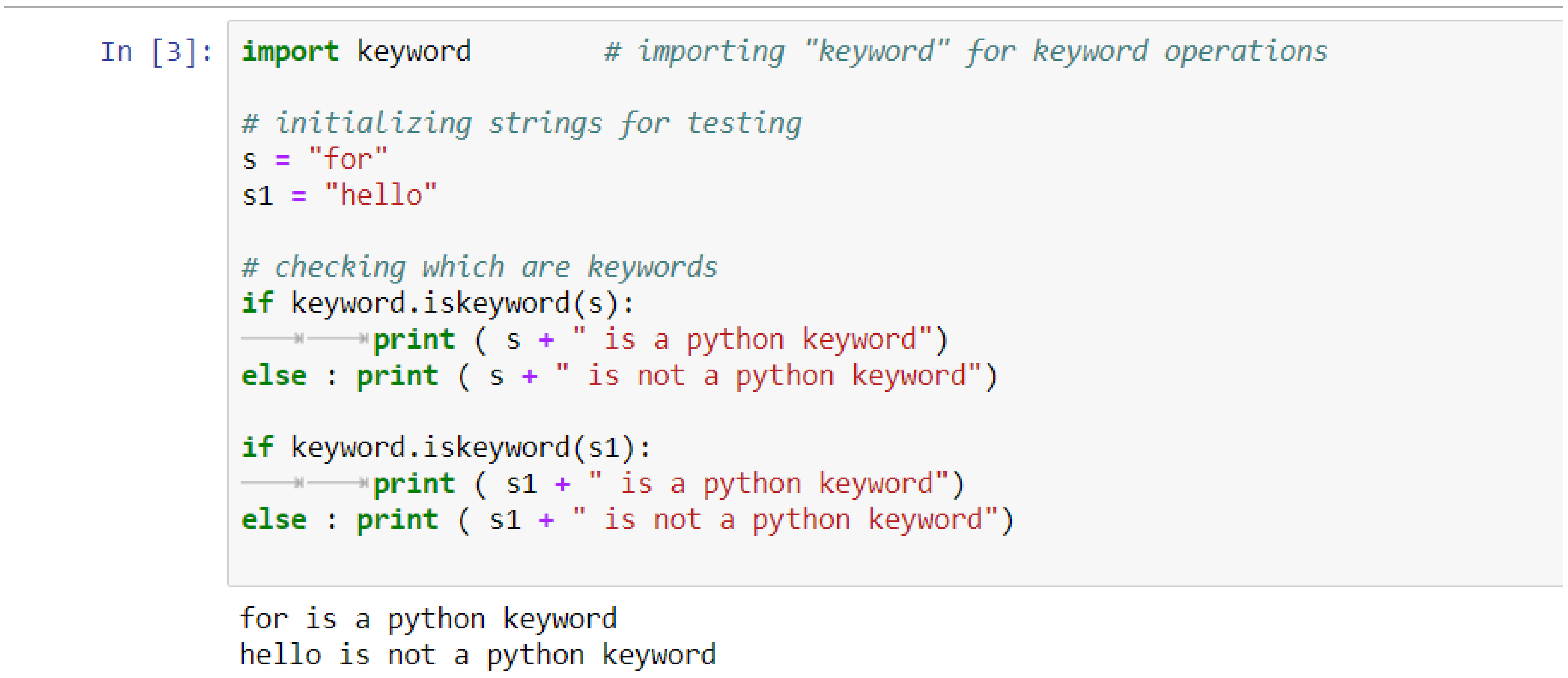
- Sometimes, remembering all the keywords can be a difficult task while assigning variable names.
- Hence a function "**kwlist()**" is provided in "keyword" module which **prints all the python keywords**.



```
In [2]: import keyword

        # printing all keywords at once using "kwlist()"
        print ("The list of keywords is : ")
        print (keyword.kwlist)

        The list of keywords is :
        ['and', 'as', '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', 'with', 'yiel
        d']
```

How to check if a string is keyword?

- A function “**iskeyword()**” checks if a string is keyword or not. Returns **true if a string is keyword, else returns false**.

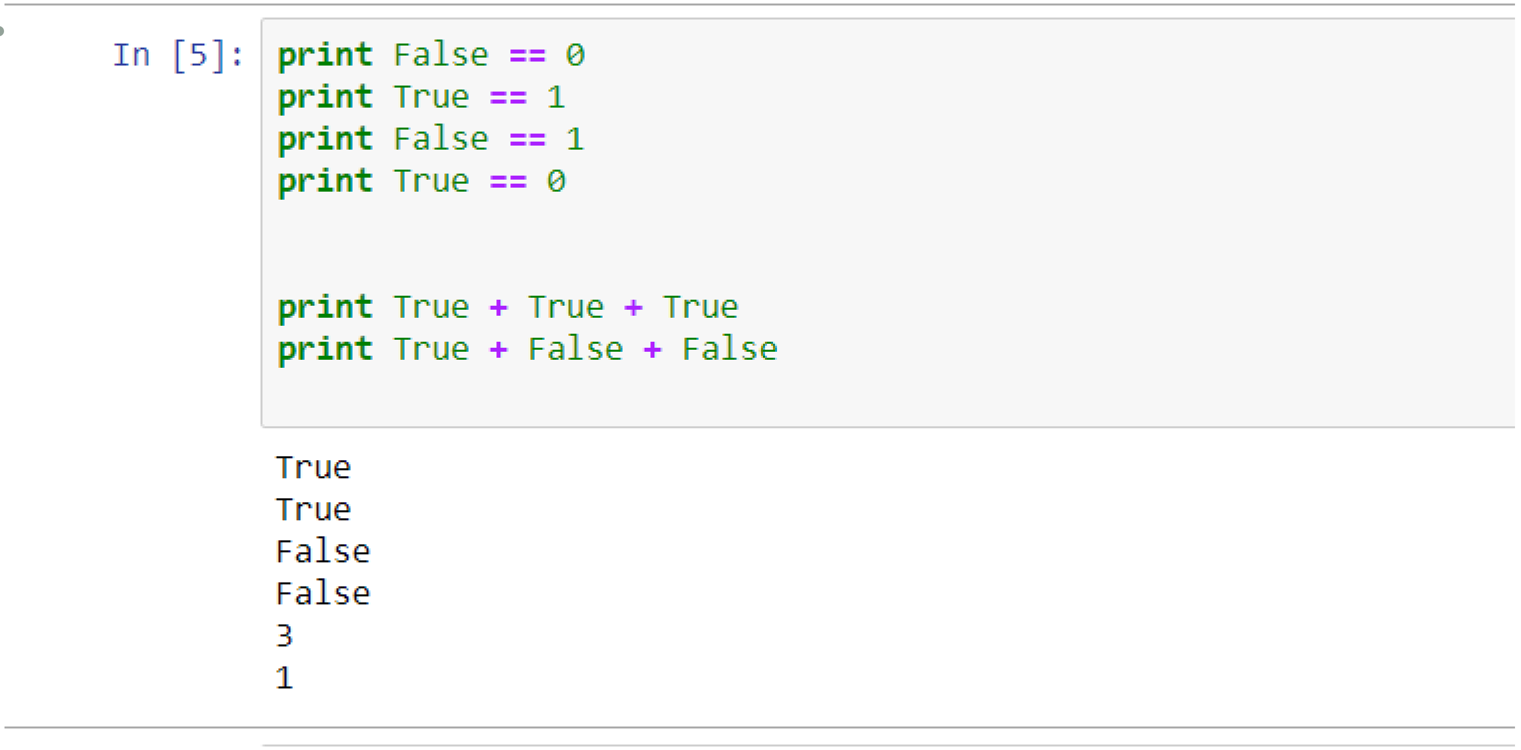


```
In [3]: import keyword         # importing "keyword" for keyword operations

        # initializing strings for testing
        s = "for"
        s1 = "hello"

        # checking which are keywords
        if keyword.iskeyword(s):
                print ( s + " is a python keyword")
        else : print ( s + " is not a python keyword")

        if keyword.iskeyword(s1):
                print ( s1 + " is a python keyword")
        else : print ( s1 + " is not a python keyword")
```

```
for is a python keyword
hello is not a python keyword
```

Detailed insight to Keywords

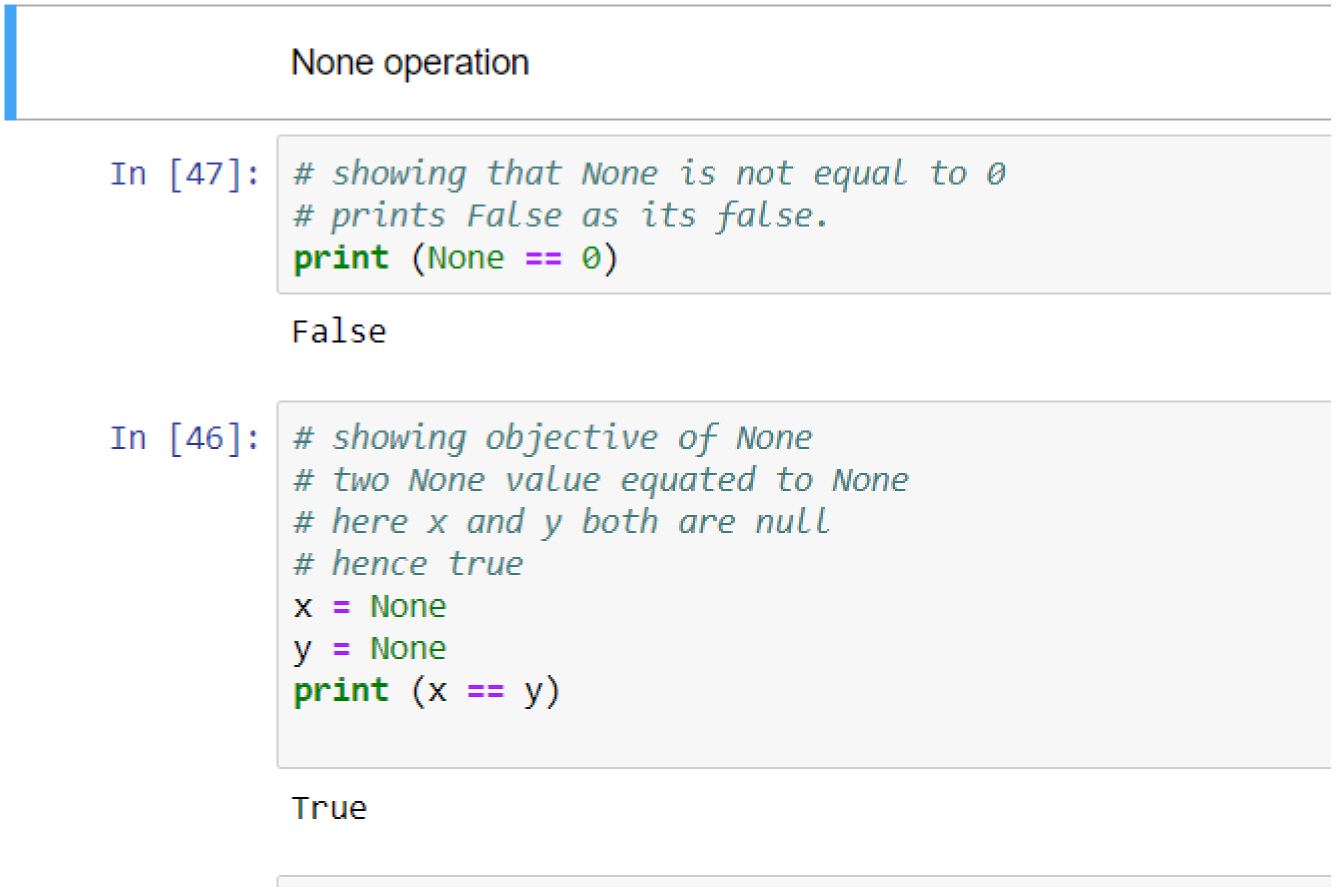
- **1. True** : This keyword is used to represent a boolean true. If a statement is true, “True” is printed.
- **2. False** : This keyword is used to represent a boolean false. If a statement is false, “False” is printed.
  True and False in python are same as **1 and 0**. Example:
-



```
In [5]: print False == 0
        print True == 1
        print False == 1
        print True == 0


        print True + True + True
        print True + False + False
```

```
True
True
False
False
3
1
```

- **3. None** : This is a special constant used to **denote a null value or a void**. **Its important to remember, 0, any empty container(e.g empty list) do not compute to None**.
  It is an object of its own datatype – NoneType. It is not possible to create multiple None objects and can assign it to variables.



None operation

In [47]:
```
# showing that None is not equal to 0
# prints False as its false.
print (None == 0)
```

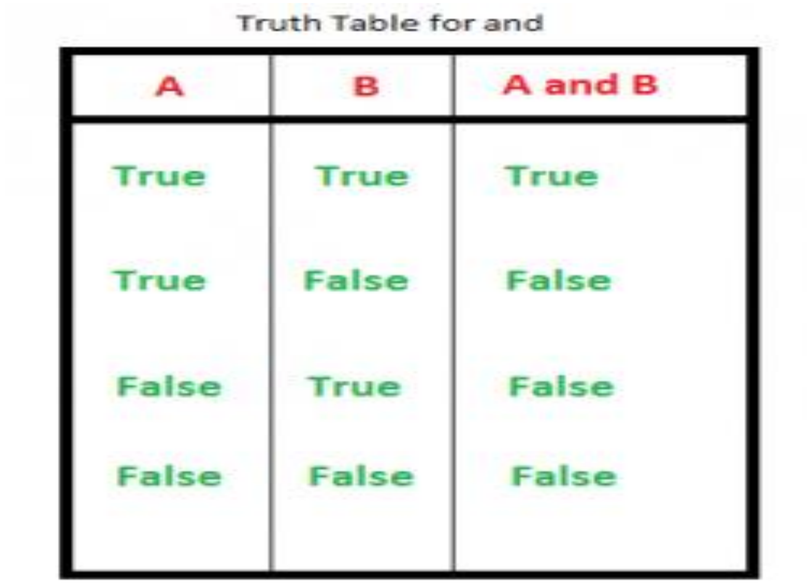
```
False
```

In [46]:
```
# showing objective of None
# two None value equated to None
# here x and y both are null
# hence true
x = None
y = None
print (x == y)
```

```
True
```

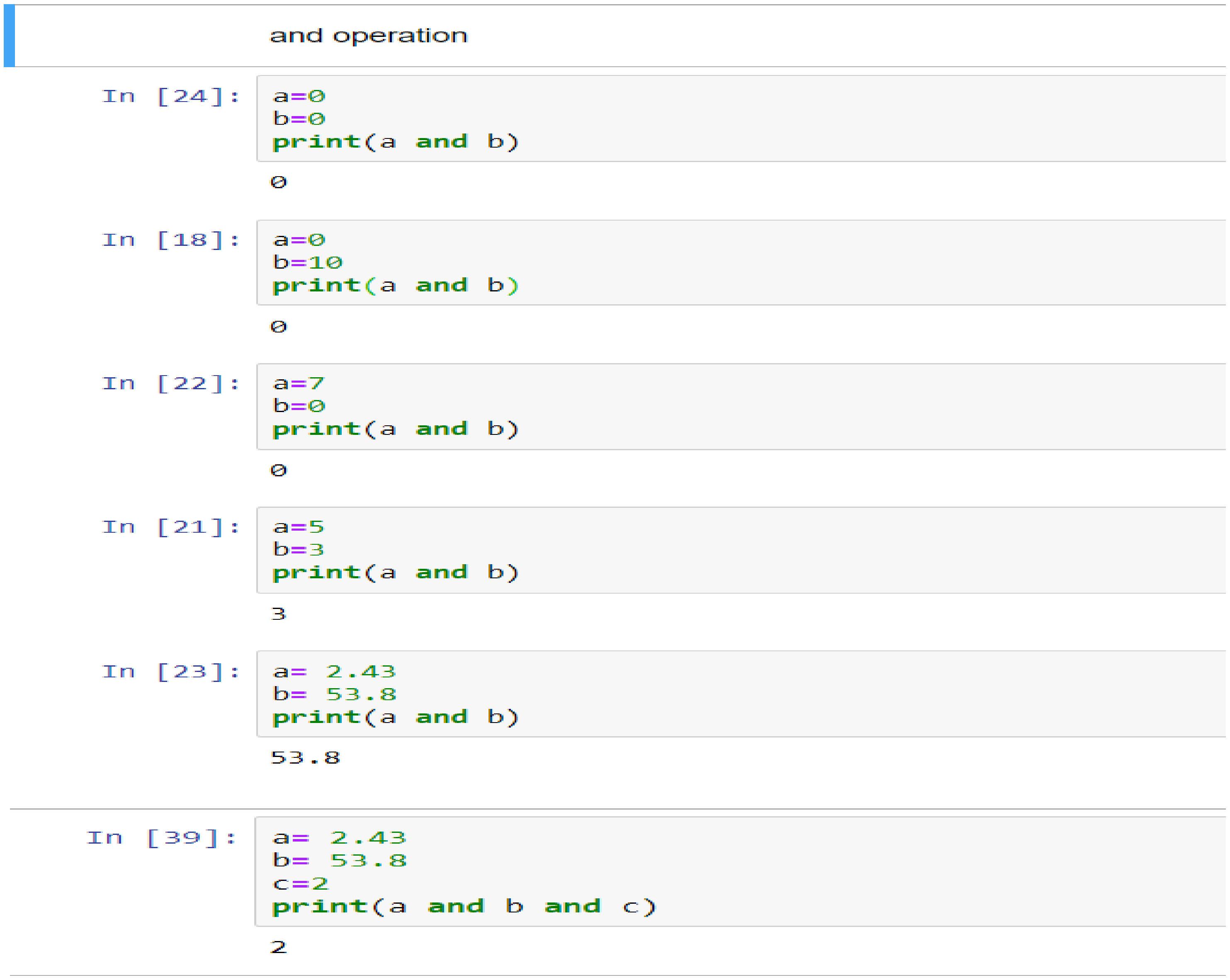
4. and :

- This a logical operator in python.
- “and” **Return the first false value .if not found return last**.
- The truth table for “and” is depicted below.



Truth Table for and

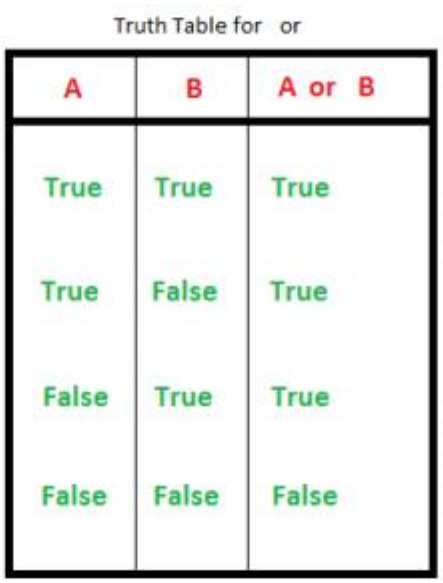
A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False



and operation

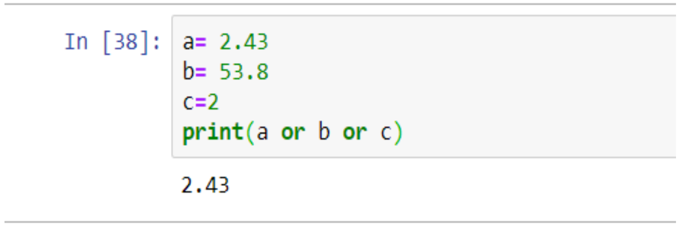
```
In [24]: a=0
         b=0
         print(a and b)
```
```
0
```
```
In [18]: a=0
         b=10
         print(a and b)
```
```
0
```
```
In [22]: a=7
         b=0
         print(a and b)
```
```
0
```
```
In [21]: a=5
         b=3
         print(a and b)
```
```
3
```
```
In [23]: a= 2.43
         b= 53.8
         print(a and b)
```
```
53.8
```
```
In [39]: a= 2.43
         b= 53.8
         c=2
         print(a and b and c)
```
```
2
```

- **5. or** : This a logical operator in python. “or” **Return the first True value.if not found return last.**The truth table for “or” is depicted below.



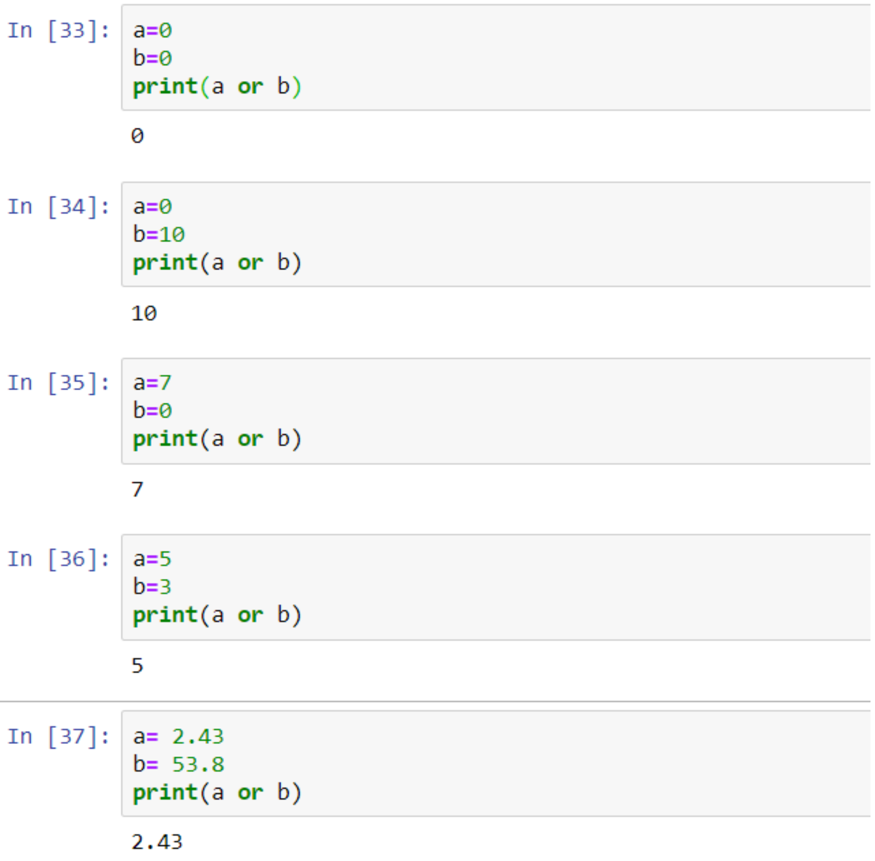
Truth Table for or

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False



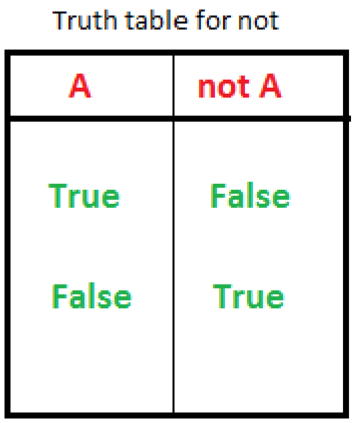
```
In [38]: a= 2.43
         b= 53.8
         c=2
         print(a or b or c)

2.43
```

or operation



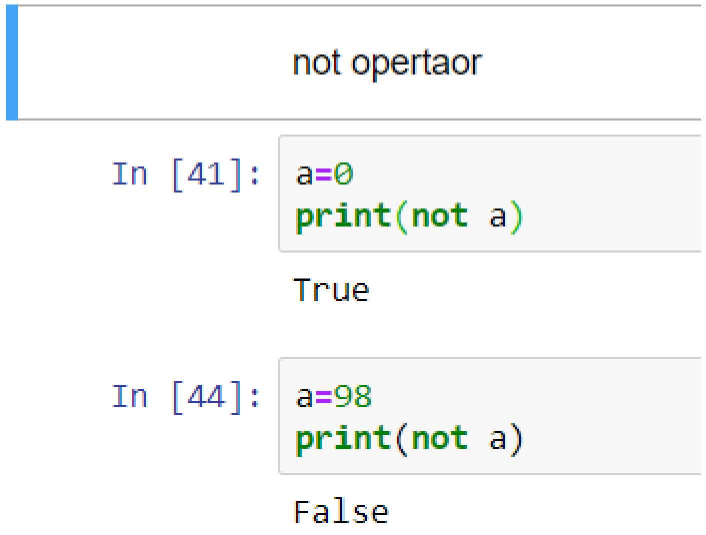
```
In [33]: a=0
         b=0
         print(a or b)

0

In [34]: a=0
         b=10
         print(a or b)

10

In [35]: a=7
         b=0
         print(a or b)

7

In [36]: a=5
         b=3
         print(a or b)

5

In [37]: a= 2.43
         b= 53.8
         print(a or b)

2.43
```

- **6. not** : This logical operator **inverts the truth value**. The truth table for "not" is depicted below.



Truth table for not

A	not A
True	False
False	True

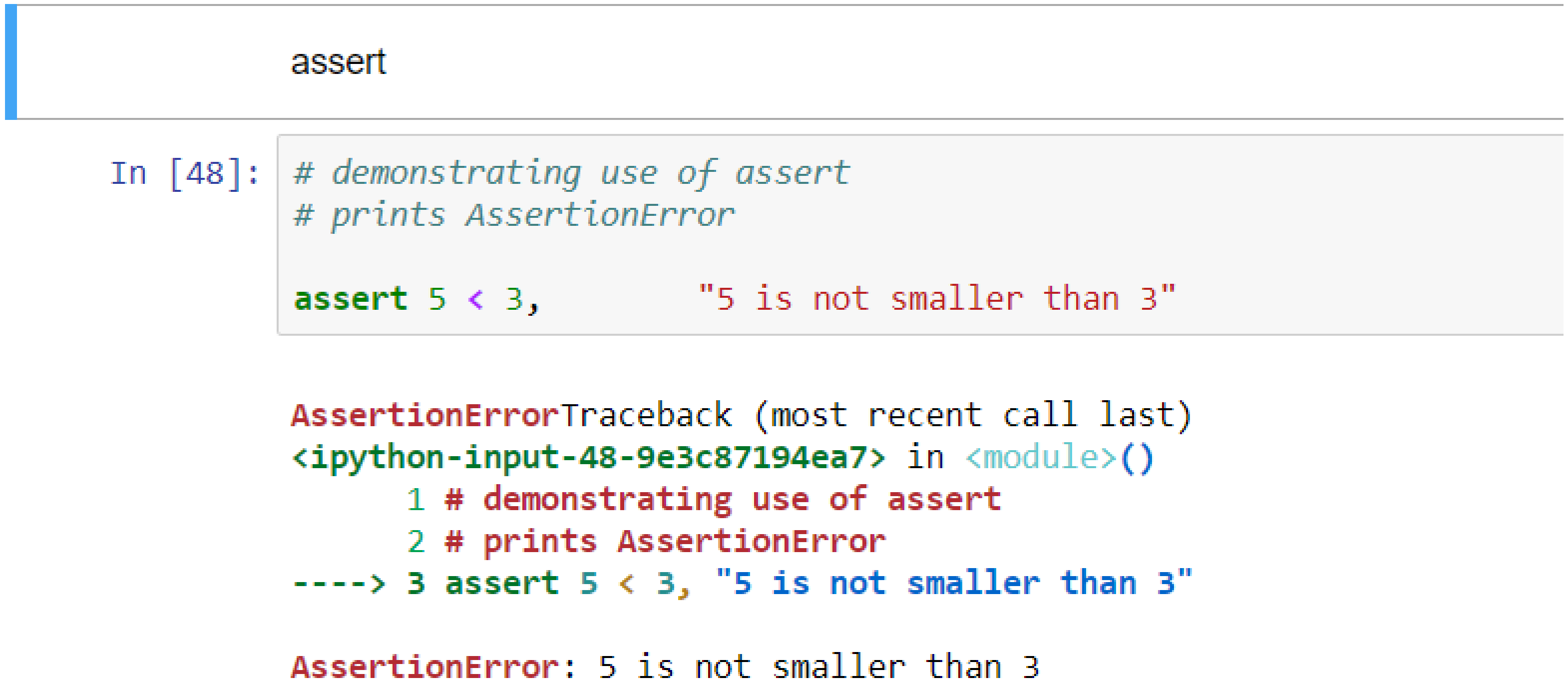


not opertaor

```
In [41]: a=0
         print(not a)

True

In [44]: a=98
         print(not a)

False
```

7. assert :

- This function is used for **debugging purposes**.
- Usually used to check the correctness of code.
- If a statement evaluated to true, nothing happens, but when it is false, “**AssertionError**” is raised .
- One can also **print a message with the error, separated by a comma**.



assert

```
In [48]: # demonstrating use of assert
         # prints AssertionError

         assert 5 < 3,        "5 is not smaller than 3"
```

```
AssertionErrorTraceback (most recent call last)
<ipython-input-48-9e3c87194ea7> in <module>()
      1 # demonstrating use of assert
      2 # prints AssertionError
----> 3 assert 5 < 3, "5 is not smaller than 3"

AssertionError: 5 is not smaller than 3
```

8. break :

- “break” is used to control the flow of loop.
- The statement is used to **break out of loop and passes the control to the statement following immediately after loop.**

9. continue :

- “continue” is also used to control the flow of code.
- The keyword **skips the current iteration of the loop**, but **does not end the loop**.

10. class :

- This keyword is used to **declare user defined classes**.

11. def :

- This keyword is used to **declare user defined functions**.

12. if :

- It is a control statement for decision making.
- **Truth expression forces control to go in “if” statement block.**

13. else :

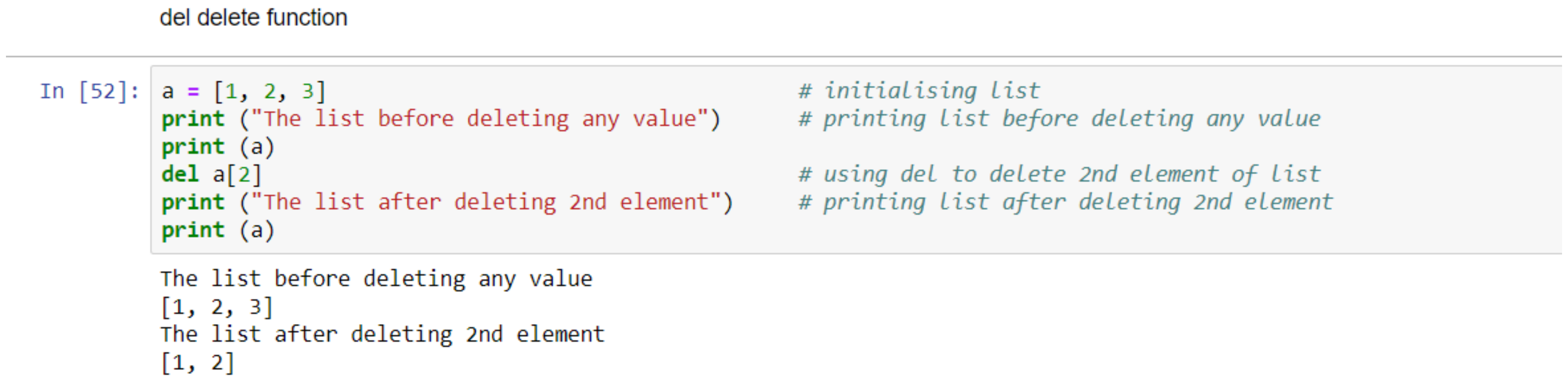
- It is a control statement for decision making.
- **False expression forces control to go in “else” statement block.**

14. elif :

- It is a control statement for decision making.
- It is short for “**else if**”

15. del :

- del is used to **delete a reference to an object**.
- Any variable or list value can be deleted using del.



del delete function

```
In [52]: a = [1, 2, 3]                                       # initialising list
         print ("The list before deleting any value")        # printing list before deleting any value
         print (a)
         del a[2]                                            # using del to delete 2nd element of list
         print ("The list after deleting 2nd element")       # printing list after deleting 2nd element
         print (a)
```

```
The list before deleting any value
[1, 2, 3]
The list after deleting 2nd element
[1, 2]
```

16. try :
- This keyword is used for exception handling**,** used to catch the errors in the code using the keyword except.
- Code in “try” block is checked, if there is any type of error, except block is executed.

17. except :
- As explained above, this works together with “try” to catch exceptions.

18. raise :
- Also used for exception handling to explicitly raise exceptions.

19. finally :
- No matter what is result of the “try” block, block termed “finally” is always executed.

20. for :
- This keyword is used to control flow and for looping.

21. while :
- Has a similar working like “for” , used to control flow and for looping.

22. pass :
- It is the null statement in python.
- Nothing happens when this is encountered.
- This is used to prevent indentation errors and used as a placeholder

23. import :
- This statement is used to include a particular module into current program.

24. from :
- Generally used with import, from is used to import particular functionality from the module imported.

25. as :
- This keyword is used to create the alias for the module imported. i.e giving a new name to the imported module.. E.g import math as mymath.

26. lambda :
- This keyword is used to make inline returning functions with no statements allowed internally.

27. return :
- This keyword is used to return from the function.

28. yield :
- This keyword is used like return statement but is used to return a generator.

29. with :
- This keyword is used to wrap the execution of block of code within methods defined by context manager.
- This keyword is not used much in day to day programming.

30. in :

- This keyword is used to check if a container contains a value.
- This keyword is also used to loop through the container.

31. is :

- This keyword is used to test object identity, i.e to check if both the objects take same memory location or not.

32. global :

- This keyword is used to define a variable inside the function to be of a global scope.

33. non-local :

- This keyword works similar to the global, but rather than global, this keyword declares a variable to point to variable of outside enclosing function, in case of nested functions.

Python Literals

- Literals can be defined as a data that is given in a variable or constant.

- Python support the following literals:

1. String literals
2. Numeric literals
3. Boolean literals
4. Special literals
5. Literal Collections

String literals

- String literals can be formed by enclosing a text in the quotes.
- We can use both single as well as double quotes for a String.

  Eg**:**

  "Aman" , '12345‘

- **Types of Strings:**
- There are two types of Strings supported in Python:

- a).**Single line String-**

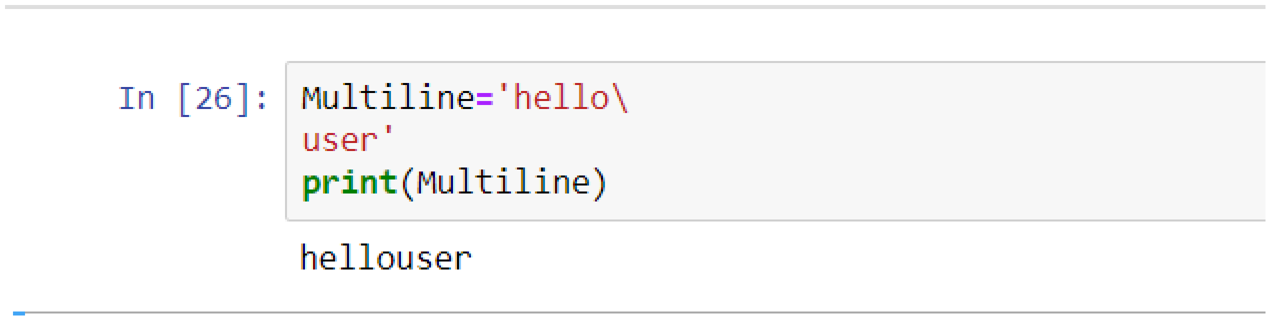
Strings that are terminated within a single line are known as Single line Strings.

Eg:

text1='hello'

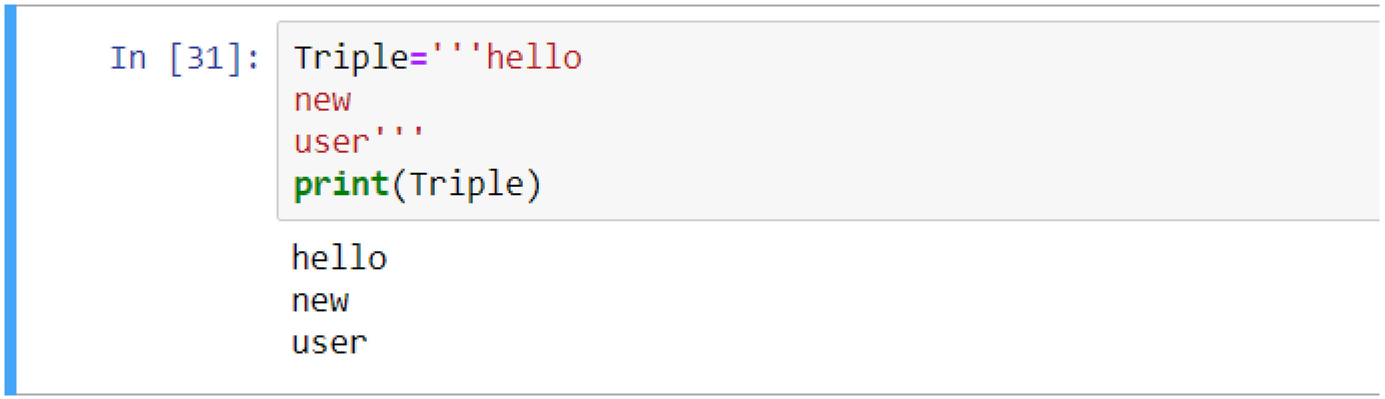
b).**Multi line String-**

- A piece of text that is spread along multiple lines is known as Multiple line String.
- There are two ways to create Multiline Strings:
- **1). Adding black slash at the end of each line.**



```
In [26]: Multiline='hello\
         user'
         print(Multiline)

         hellouser
```

- **2).Using triple quotation marks:**



```
In [31]: Triple='''hello
         new
         user'''
         print(Triple)

         hello
         new
         user
```

Numeric literals

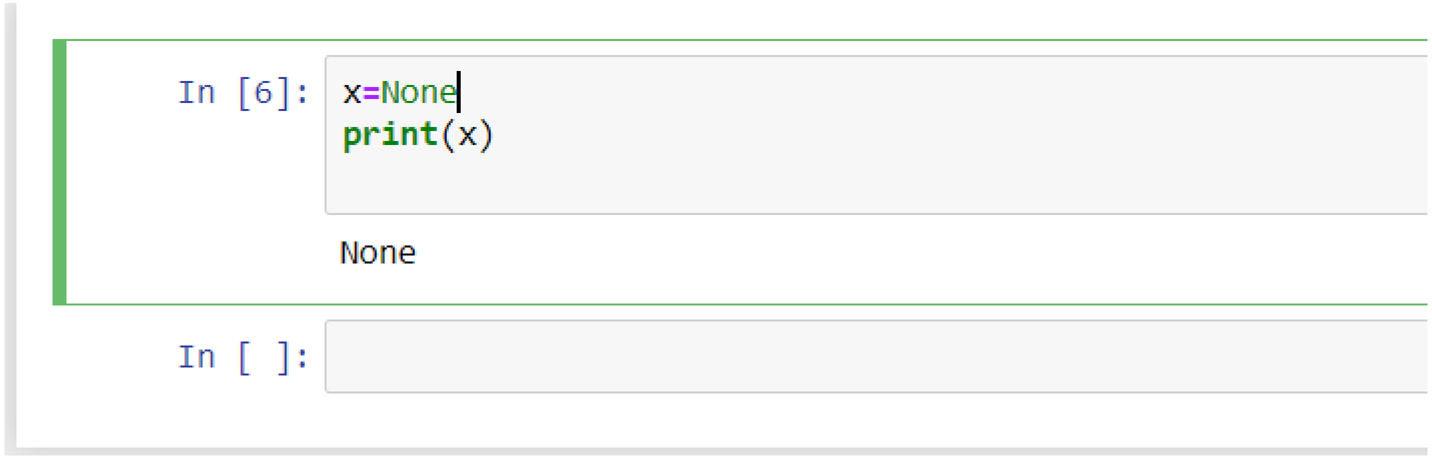
Int(signed integers)	Long(long integers)	float(floating point)	Complex(complex)
Numbers(can be both positive and negative) with no fractional part.eg: 100	Integers of unlimited size followed by lowercase or uppercase L eg: 87032845L	Real numbers with both integer and fractional part eg: -26.2	In the form of a+bj where a forms the real part and b forms the imaginary part of complex number. eg: 3.14j

Boolean literals:

- A Boolean literal can have any of the two values: True or False.

Special literals.

- Python contains one special literal i.e., *None*.
- None is used to specify to that field that is not created. It is also used for end of lists in Python.



```
In [6]: x=None
        print(x)

        None

In [ ]:
```

Literal Collections.

- Collections such as tuples, lists and Dictionary are used in Python.

Python Operators

The operator can be defined as a symbol which is responsible for a particular operation between two operands.

Python provides a variety of operators described as follows.

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic operators

- Arithmetic operators are used to perform arithmetic operations between two operands.

Operator	Description
+ (Addition)	It is used to add two operands. For example, if a = 20, b = 10 => a+b = 30
- (Subtraction)	It is used to subtract the second operand from the first operand. If the first operand is less than the second operand, the value result negative. For example, if a = 20, b = 10 => a ? b = 10
/ (divide)	It returns the quotient after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a/b = 2
*** (Multiplication)**	It is used to multiply one operand with the other. For example, if a = 20, b = 10 => a * b = 200
% (reminder)	It returns the reminder after dividing the first operand by the second operand. For example, if a = 20, b = 10 => a%b = 0
**** (Exponent)**	It is an exponent operator represented as it calculates the first operand power to second operand.
// (Floor division)	It gives the floor value of the quotient produced by dividing the two operands.

Comparison operator

- Comparison operators are used to comparing the value of the two operands and returns boolean true or false accordingly.

Operator	Description
==	If the value of two operands is equal, then the condition becomes true.
!=	If the value of two operands is not equal then the condition becomes true.
<=	If the first operand is less than or equal to the second operand, then the condition becomes true.
>=	If the first operand is greater than or equal to the second operand, then the condition becomes true.
<>	If the value of two operands is not equal, then the condition becomes true.
>	If the first operand is greater than the second operand, then the condition becomes true.
<	If the first operand is less than the second operand, then the condition becomes true.

Python assignment operators

- The assignment operators are used to assign the value of the right expression to the left operand.

Operator	Description
=	It assigns the the value of the right expression to the left operand.
+=	It increases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a+ = b will be equal to a = a+ b and therefore, a = 30.
-=	It decreases the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 20, b = 10 => a- = b will be equal to a = a- b and therefore, a = 10.
*=	It multiplies the value of the left operand by the value of the right operand and assign the modified value back to left operand. For example, if a = 10, b = 20 => a* = b will be equal to a = a* b and therefore, a = 200.
%=	It divides the value of the left operand by the value of the right operand and assign the reminder back to left operand. For example, if a = 20, b = 10 => a % = b will be equal to a = a % b and therefore, a = 0.
**=	a**=b will be equal to a=a**b, for example, if a = 4, b =2, a**=b will assign 4**2 = 16 to a.
//=	A//=b will be equal to a = a// b, for example, if a = 4, b = 3, a//=b will assign 4//3 = 1 to a.