



SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai



DEPARTMENT OF CSE (IoT & CYBER SECURITY INCLUDING BLOCKCHAIN TECHNOLOGY)



19IT103 – COMPUTATIONAL THINKING AND PYTHON PROGRAMMING

❖ A readable, dynamic, pleasant, flexible, fast and powerful language

UNIT II DATA TYPES, EXPRESSIONS, STATEMENTS

- Python interpreter and interactive mode, debugging; values and types: int, float, boolean, string, and list; variables, **expressions, statements, tuple assignment**, precedence of operators, comments; Illustrative programs: exchange the values of two variables, circulate the values of n variables, distance between two points.

Recap

- Variables
- Object References
- Rules for Naming Identifier

Expressions

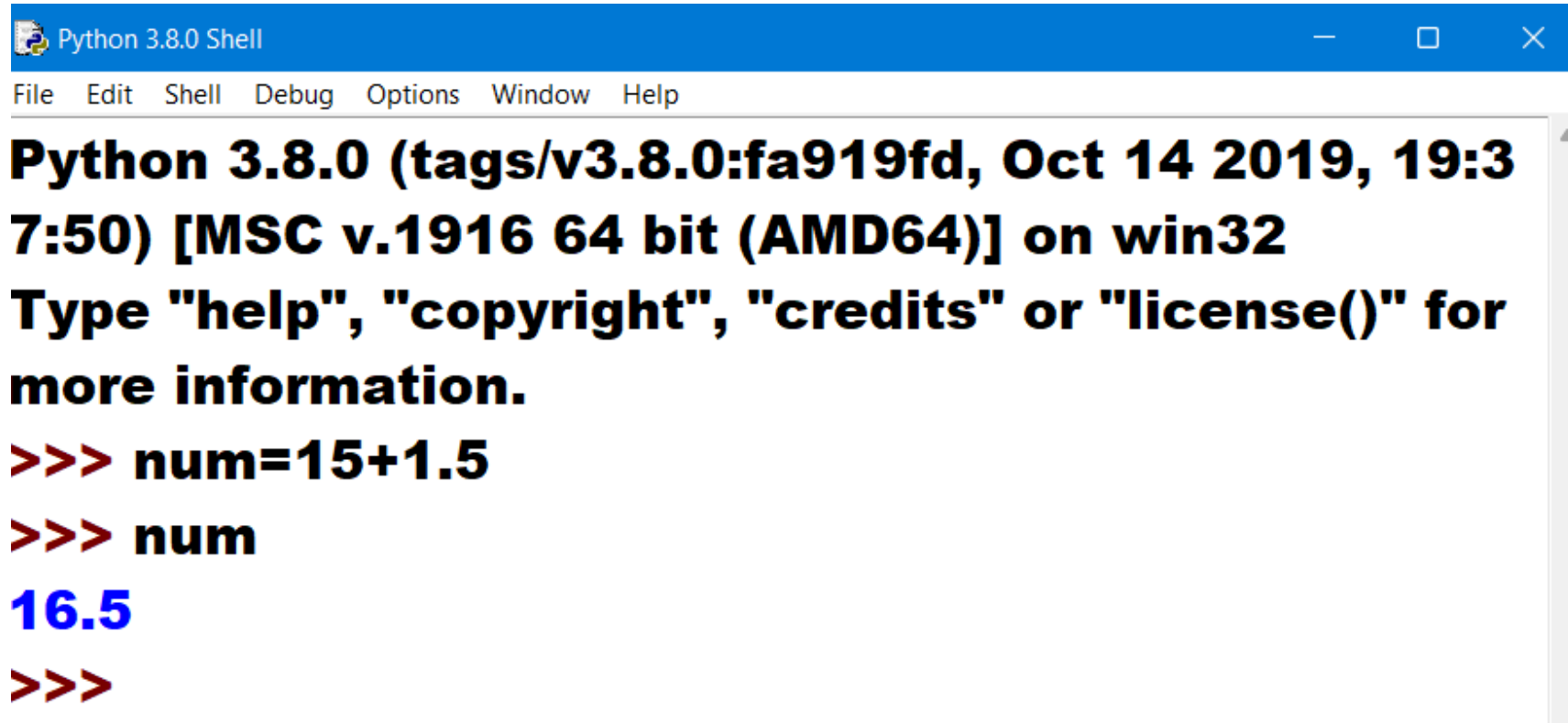
- An expression is a **combination of operators and operands** that is interpreted to produce some other value.
- An expression is **evaluated as per the precedence** of its operators.
- If there is more than one operator in an expression, their precedence decides which operation will be performed first.

Expressions

- Types of Expressions:
 - Constant Expressions
 - Arithmetic Expressions
 - Integral Expressions
 - Floating Expressions
 - Relational Expressions
 - Logical Expressions
 - Bitwise Expressions
 - Combinational Expressions

Constant Expressions

- These are the expressions that have **constant values** only.



```
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> num=15+1.5
>>> num
16.5
>>>
```

Arithmetic Expressions

- An arithmetic expression is a **combination of numeric values, operators, and sometimes parenthesis.**
- The **result** of this type of expression is also a **numeric value.**
- The operators used in these expressions are **arithmetic operators** like addition, subtraction, etc.

Arithmetic Expressions

```
>>> num1=40
```

```
>>> num2=30
```

```
>>> add=num1+num2
```

```
>>> add
```

```
70
```

```
>>> sub=num1-num2
```

```
>>> sub
```

```
10
```

```
>>> prod=num1 * num2
```

```
>>> prod
```

```
1200
```

```
>>> div=num1/num2
```

```
>>> div
```

```
1.3333333333333333
```

```
>>>
```


Integral Expressions

- These are the kind of expressions that **produce only integer results** after all computations and type conversions.

```
>>> num1=12
>>> num2=32.0
>>> result=num1+int(num2)
>>> result
44
>>> |
```

Floating Expressions

- These are the kind of expressions which **produce floating point numbers** as result after all computations and type conversions.

```
>>> num1=13
>>> num2=3
>>> print(num1/num2)
4.333333333333333
>>> |
```

Relational Expressions

- In these types of expressions, **arithmetic expressions** are written on **both sides of relational operator** ($>$, $<$, $>=$, $<=$).
- Those arithmetic expressions are evaluated first, and then compared as per relational operator and produce a **boolean output** in the end.
- These expressions are also called **Boolean expressions**.

Relational Expressions

```
>>> num1=21
```

```
>>> num2=13
```

```
>>> num3=40
```

```
>>> expr=(num1+num2)>=(num3-num2)
```

```
>>> expr
```

```
True
```

```
>>>
```

Logical Expressions

- These are kinds of expressions that result in either **True or False**. It basically specifies **one or more conditions**.

```
Python 3.8.0 Shell
File Edit Shell Debug Options Window Help
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:37:50) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> num1=(10==9)
>>> num2=(7>5)
>>> print(num1 and num2)
False
>>> print(num1 or num2)
True
>>> print(not num1)
True
>>> |
```

Bitwise Expressions

- These are the kind of expressions in which **computations** are performed at **bit level**.

```
>>> num=12
>>> print(num>>2)
3
>>> print(num<<2)
48
>>> |
```

Combinational Expressions

- Use **different types of expressions** in a single expression, and that will be termed as combinational expressions.

```
>>> num1=10
>>> num2=15
>>> print(num1+(num2>>1))
17
>>> |
```

Combinational Expressions

- Use **different types of expressions** in a single expression, and that will be termed as combinational expressions.

```
>>> num1=10
>>> num2=15
>>> print(num1+(num2>>1))
17
>>> |
```


Statements

- A statement is a unit of code that the Python interpreter can execute.
- Statements represent an action or command
 - Eg: print, assignment statement
- The important difference is that an expression has a value; a statement does not.
- Statements on the other hand, are everything that can make up a line (or several lines) of Python code.
- Expressions are statements as well.

Statements vs Expressions

Expressions	Statements
An expression evaluates to a value	Statements represent an action or command
The evaluation of a statement does not change state	The execution of a statement changes state
Evaluation of an expression always Produces or returns a result value.	Execution of a statement may or may not produces or displays a result value, it only does whatever the statement says.
Example: <code>>>> a + 16</code> <code>>>> 20</code>	Example: <code>>>> x = 3</code> <code>>>> print(x)</code> Output: 3

Tuple Assignment

- Tuple is **sequence** data type.
- Initialise or create a tuple in various ways.
- The process of **assigning values** to a tuple is known as **packing**.
- The **unpacking** or tuple assignment is the process that **assigns** the values on the **right-hand side to the left-hand side variables**.

Tuple Packing (Creating Tuples)

- Tuple can contain all elements of the same data type as well as of mixed data types as well.

```
>>>tup = (22, 33, 5, 23)
```

```
>>>tup
```

```
(22, 33, 5, 23)
```

Tuple Packing (Creating Tuples)

- Tuple with mixed data type

```
>>>tup2 = ('hi', 11, 45.7)
```

```
>>>tup2
```

```
('hi', 11, 45.7)
```

Tuple Packing (Creating Tuples)

- Tuple with a tuple as an element

```
>>>tup3 = (55, (6, 'hi'), 67)
```

```
>>>tup3
```

```
(55, (6, 'hi'), 67)
```

```
>>> tup3 = (55, (6, 'hi'), 67)
```

```
>>> tup3[1][1]
```

```
'hi'
```

Tuple Packing (Creating Tuples)

- Tuple with a list as an element

```
>>>tup3 = (55, [6, 9], 67)
```

```
>>>tup3
```

```
(55, [6, 9], 67)
```

```
>>> tup3 = (55, [6, 9], 67)
```

```
>>> tup3[1][0]
```

```
6
```

```
>>> |
```

Tuple Packing (Creating Tuples)

- If there is only a single element in a tuple we should end it with a comma.
- Since writing, just the element inside the parenthesis will be considered as an integer.

```
>>> tup=(90)
>>> type(tup)
<class 'int'>
>>> tup=(90,)
>>> type(tup)
<class 'tuple'>
>>> |
```


Tuple Packing (Creating Tuples)

- If there is only a single element in a tuple we should end it with a comma.
- Since writing, just the element inside the parenthesis will be considered as an integer.

```
>>> tup=(90)
>>> type(tup)
<class 'int'>
>>> tup=(90,)
>>> type(tup)
<class 'tuple'>
>>> |
```

Tuple Packing (Creating Tuples)

- If you write any sequence separated by commas, python considers it as a tuple.

```
>>> seq = 22, 4, 56
```

```
>>> print(seq)
```

```
(22, 4, 56)
```

```
>>> type(seq)
```

```
<class 'tuple'>
```

```
>>> |
```

Tuple Assignment (Unpacking)

- Unpacking or tuple assignment is the process that assigns the values on the right-hand side to the left-hand side variables.

```
>>>(n1, n2) = (99, 7)
```

```
>>>print(n1)
```

```
99
```

```
>>>print(n2)
```

```
7
```

Tuple Assignment (Unpacking)

```
>>>tup1 = (8, 99, 90, 6.7)

>>>(roll no., english, maths, GPA) = tup1

>>>print(english)

99

>>>print(roll no.)

8

>>>print(GPA)

6.7

>>>print(maths)

90
```

Tuple Assignment (Unpacking)

```
>>> (num1, num2, num3, num4, num5) = (88, 9.8, 6.8, 1)
```

```
#this gives an error as the variables on the left are more than the  
number of elements in the tuple
```

```
ValueError: not enough values to unpack
```

```
(expected 5, got 4)
```