Simple Operations

Instructor: Bulend Hoca

CLARUSWAY©
WAY TO REINVENT YOURSELF

Arithmetic Operations

Arithmetic Operations

Operator	Description	Example
`+`	Addition operator	`100 + 45 = 145`
`-`	Subtraction operator	`500 - 65 = 435`
`*`	Multiplication operator	`25 * 4 = 100`
`/`	Float Division Operator	`10 / 2 = 5.0`
`//`	Integer Division Operator	`11 // 2 = 5`
`**`	Exponentiation Operator	`5 ** 3 = 125`
`%`	Remainder Operator	`10 % 3 = 1`

Arithmetic Operations

- Interactive question :

```
print(11-7)
print(4 + 11.0)
print('11 - 7')
print('4' + 4)
```

What is the output?

Arithmetic Operations

- The output :

```python
print(11-7)
print(4 + 11.0)
print('11 - 7')
print('4' + 4)

```

```
4
15.0
11 - 7
Traceback (most recent call last):
  File "code.py", line 5, in <module>
    print('4'+ 4)
TypeError: can only concatenate str (not "int") to str
```

Arithmetic Operations

- Interactive question :

```
num1, num2 = 81, 55
num3 = num1 - num2
print(num3)
```

What is the output?

Arithmetic Operations

- Interactive question :

```
num1, num2 = 81, 55
num3 = num1 - num2
print(num3)
```

Arithmetic Operations

- The output :

```
num1, num2 = 81, 55
num3 = num1 - num2
print(num3)
```

Output

```
26
```

Arithmetic Operations

- Interactive question :

```
print(11 % 2)  # remainder of this division is 1
               # it means 11 is an odd number
print((4 * 5) / 2)  # parentheses are used as in normal math operations
```

What is the output?

Arithmetic Operations

- The output :

```
print(11 % 2)  # remainder of this division is 1
               # it means 11 is an odd number
print((4 * 5) / 2)  # parentheses are used as in normal math operations

```

```
1

10.0
```

Arithmetic Operations

- Interactive question :

```python
print(2 ** 3)  # 2 to the power of 3
print(3 ** 2)  # square of 3
a = 2
b = 8
print((a * b) ** 0.5)  # square root
```

What is the output?

Arithmetic Operations

- The output :

```python
print(2 ** 3)  # 2 to the power of 3
print(3 ** 2)  # square of 3
a = 2
b = 8
print((a * b) ** 0.5)  # square root

```

```
8
9
4.0
```

Arithmetic Operations

Tips:

- `Variable math operator = number` gives the same result as `Variable = Variable math operator number`.
- `Variable += number` gives the same result as `Variable = Variable + number`.

- `-=` decrements the variable in place,
- `+=` increment the variable in place,
- `*=` multiply the variable in place,
- `/=` divide the variable in place,
- `//=` floor divide the variable in place,
- `%=` returns the modulus of the variable in place,
- `**=` raise to power in place.

```
x += 3 ⇐⇒ x = x + 3
x *= 3 ⇐⇒ x = x * 3
x **= 3 ⇐⇒ x = x ** 3
```

Arithmetic Operations

1. parentheses : `()`
2. power : `**`
3. unary minus : `-3`
4. multiplication and division : `*`, `/`
5. addition and subtraction : `+`, `-`

Arithmetic Operations

- Interactive question :

```
a = (1 + 3 ) ** (2 ** (1 * 2 / 2) / 2)
print(a)
```

What is the output?

Arithmetic Operations

- The output :

```
a = (1 + 3 ) ** (2 ** (1 * 2 / 2) / 2)
print(a)
```

```
4.0
```

Escape Sequences

\n

\t

\b

Escape Sequences (review)

Python ignores any character which comes immediately after `\` .

- `\n` : means new line,
- `\t` : means `tab` mark,
- `\b` : means backspace. It moves the cursor one character to the left.

Escape Sequences

- **Let's** take a closer look at the escape sequences through the examples.

```
print('C:\\north pole\noise_penguins.txt')
print('----------')
print('first', 'second', 'third', sep='\t')
```

What is the output? Try to guess in your mind...

Escape Sequences

- **Let's** take a closer look at the escape sequences through the examples.

```
print('C:\\north pole\noise_penguins.txt')
print('----------')
print('first', 'second', 'third', sep='\t')
```

```
C:\north pole
oise_penguins.txt
----------
first	second	third
```

Escape Sequences, Quiz

- **Let's** take a closer look at the escape sequences through the examples.

```
print('we are', '\boosting', 'our', '\brotherhood')
print('it\'s essential to learn Python\'s libraries in IT World')
```

What is the output? Try to guess in your mind...

Escape Sequences, Quiz

- **Let's** take a closer look at the escape sequences through the examples.

```
print('we are', '\boosting', 'our', '\brotherhood')
print('it\'s essential to learn Python\'s libraries in IT World')
```

```
we areoosting ourrotherhood
it's essential to learn Python's libraries in IT World
```

Boolean Operations

CLARUSWAY©
WAY TO REINVENT YOURSELF

Table of Contents

Boolean Logic Expressions

Order of Priority

Truth Values of Logic Statements

Boolean Logic Expressions

not

and

or

Boolean Logic Expressions

- There are three built-in operators in Python :

`and`

It evaluates all expressions and returns the **last** expression if **all** expressions are evaluated `True`. Otherwise, it returns the **first** value that evaluated `False`.

`or`

It evaluates the expressions left to right and returns the first value that evaluated `True` or the last value (if none is `True`).

`not`

It evaluates the expression that follows it as the opposite of the truth. eg. `not True` means `False`

Boolean Logic Expressions

- Table of Logic Expressions in Python :

Value1	Logic	Value2	Returns
True	and	True	True
True	and	False	False
False	and	False	False
False	and	True	False
True	or	True	True
True	or	False	True
False	or	False	False
False	or	True	True

It's better to keep this table in mind.

Order of Priority

CLARUSWAY©
WAY TO REINVENT YOURSELF

Order of Priority

- Here are the operators **in order** of their **priorities** :

1. `not`
2. `and`
3. `or`

Order of Priority

- It is important to remember that, logical operators have a different priority and it has an effect on the order of evaluation.
- Here are the operators in order of their priorities :

1. not
2. and
3. or

```
bool_var = False and not True
print(bool_var)
```

Order of Priority

- It is important to remember that, logical operators have a different priority and it has an effect on the order of evaluation.
- Here are the operators in order of their p

Firstly evaluated.
The result = `False`

1. `not`
2. `and`
3. `or`

```
bool_var = False and not True
print(bool_var)
```

Order of Priority

- It is important to remember that, logical operators have a different priority and it has an effect on the order of evaluation.
- Here are the op er of their p

1. not
2. and
3. or

Secondly evaluated.
`False and False = False`

Firstly evaluated.
The result = `False`

```
bool_var = False and not True
print(bool_var)
```

Order of Priority

- It is important to remember that, logical operators have a different priority and it has an effect on the order of evaluation.
- Here are the op er of their p

Secondly evaluated.
`False and False = False`

Firstly evaluated.
The result = `False`

1. `not`
2. `and`
3. `or`

```
bool_var = False and not True
print(bool_var)
```

```
False
```

Order of Priority

💡Tips:

- Note that `and` and `or` return one of its operands, not necessarily a `bool` type. But `not` always returns `bool` type.

```
print(1 and 0)
print(not 0)
```

What is the output? Try to guess in your mind...

Order of Priority

Tips:

- Note that `and` and `or` return one of its operands, not necessarily a `bool` type. But `not` always returns `bool` type.

```
print(1 and 0)
print(not 0)
```

```
0
True
```

Truth Values of Logic Statements

The following values are considered False, in that they evaluate to **`False`** when applied to a boolean operator:

- `None`.
- Zero of any numeric type: `0, 0.0, 0j`
- Empty sequences and collections: `'', [], {}`.
- Other than above values, any remaining value is evaluated as `True`.

```
print(3 and "I am doing good!")
print(3 or "I am doing good!")
```

```
I am doing good!
3
```

any & all

`all`(*`iterable`*)

Return `True` if **all elements** of the *iterable* are true (or if the iterable is empty).

`any`(*`iterable`*)

Return `True` if **any element** of the *iterable* is true. If the iterable is empty, return `False`.

any & all

Truth table

```
+-----------------------------------------+---------+---------+
|                                         |   any   |   all   |
+-----------------------------------------+---------+---------+
| All Truthy values                       |  True   |  True   |
+-----------------------------------------+---------+---------+
| All Falsy values                        |  False  |  False  |
+-----------------------------------------+---------+---------+
| One Truthy value (all others are Falsy) |  True   |  False  |
+-----------------------------------------+---------+---------+
| One Falsy value (all others are Truthy) |  True   |  False  |
+-----------------------------------------+---------+---------+
| Empty Iterable                          |  False  |  True   |
+-----------------------------------------+---------+---------+
```

any & all

```
print(any([False, False, False]))
#??

print(any([False, False, True]))
#??

print(all([False, True, True]))
#??
print(all([True, True, True]))
#??
```

any & all

```
print(any([False, False, False]))
#False
print(any([False, False, True]))
#True

print(all([False, True, True]))
#False
print(all([True, True, True]))
#True
```