Basic Programming

Lesson 02

Operators

Arithmetic Operators

Operator	Description	Example
+	Addition	1 + 1 = 2
-	Subtraction	10 - 1 = 9
*	Multiplication	3 * 5 = 15
/	Division	10 / 5 = 2
%	Modulus (remainder after division)	11 % 5 = 1
**	Exponent	3**2 = 9
//	Floor division	11 // 5 = 2

Relational Operators

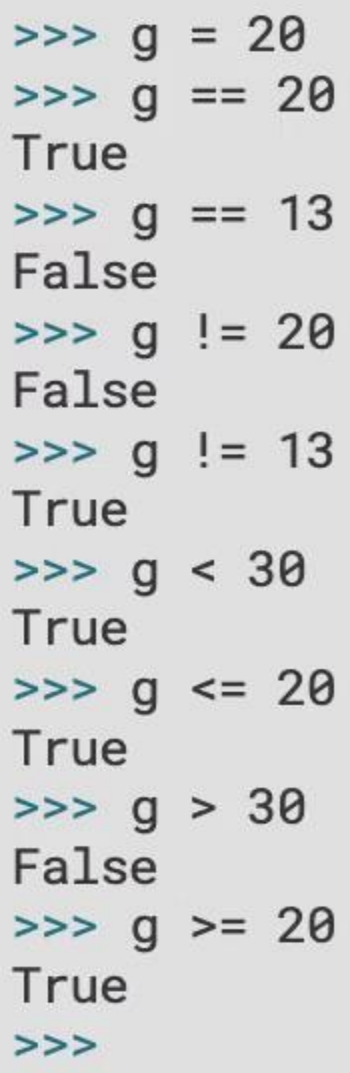
Operator	Meaning
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to
is	Object identity
is not	Negated object identity

Value Equality

==

True

Relational Operators



```
>>> g = 20
>>> g == 20
True
>>> g == 13
False
>>> g != 20
False
>>> g != 13
True
>>> g < 30
True
>>> g <= 20
True
>>> g > 30
False
>>> g >= 20
True
>>>
```

Boolean Operators

Operator	Code Example	What It Determines
or	x or y	Either x or y is True
and	x and y	Both x and y are True
not	not x	x is not True

Control Flow

Conditional statement

Branch execution based on the value of an expression

If-statement Syntax

```
if expression:
    block
```

While-loops

While-loops

```
while expression:
    block
```

converted to boolean

Relational Operators

```
>>> while True:
...     pass
...
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyboardInterrupt
>>>
```

break

Many languages support a loop ending in a predicate test

C, C++, C#, and Java have do-while

Python requires you to use `while True` and `break`

`break` jumps out of the inner-most executing loop to the line immediately after it

Break

```
>>> while True:
...         response = input()
...         if int(response) % 7 == 0:
...                 break
...
12
67
34
28
>>>
```

For-loops

For-loops

```
for item in iterable:
    ...body...
```

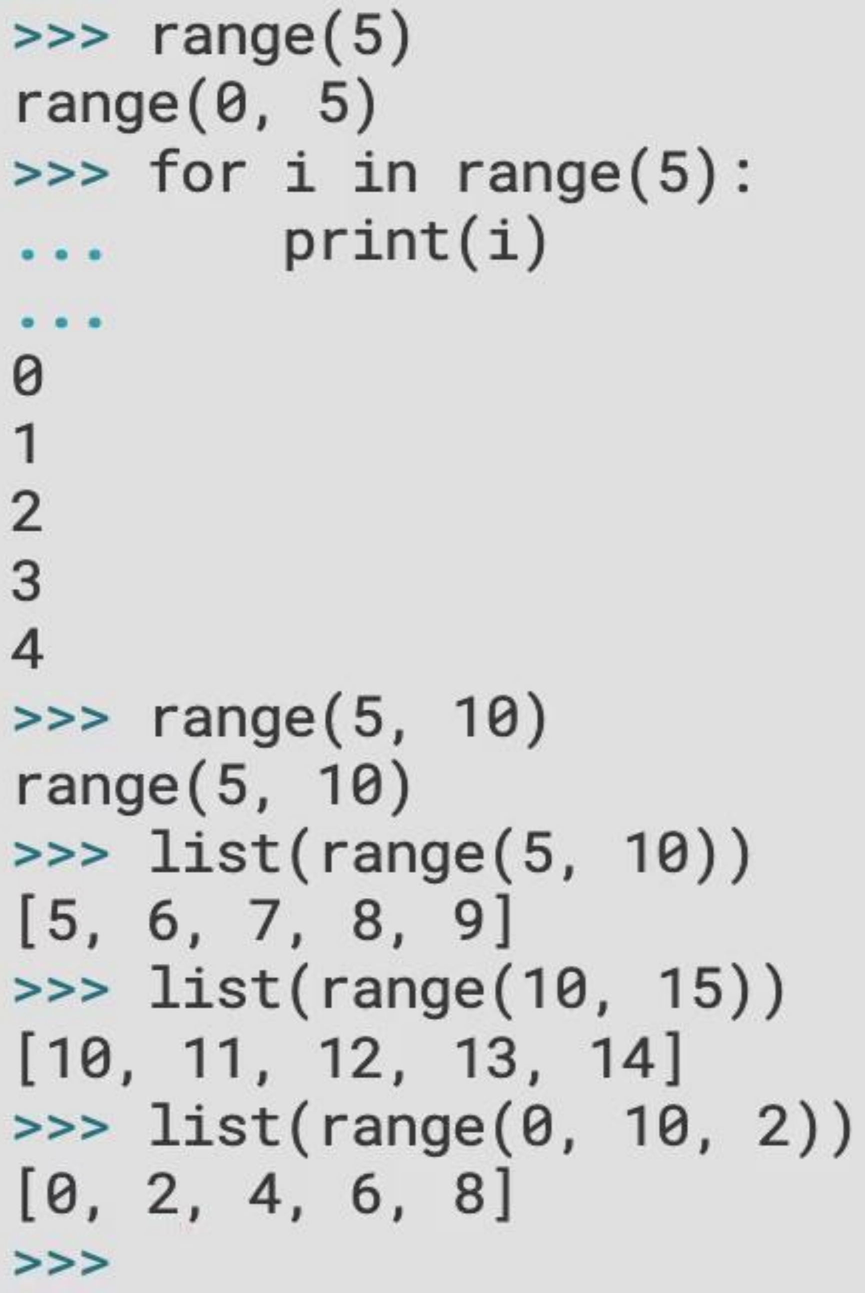
For-loop

```
>>> cities = ["London", "New York", "Paris", "Oslo", "Helsinki"]
>>> for city in cities:
...     print(city)
...
London
New York
Paris
Oslo
Helsinki
>>> colors = {'crimson': 0xdc143c, 'coral': 0xff7f50, 'teal': 0x008080}
>>> for color in colors:
...     print(color, colors[color])
...
crimson 14423100
coral 16744272
teal 32896
>>>
```

Range

Range

Sequence representing an arithmetic progression of integers



```
>>> range(5)
range(0, 5)
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
>>> range(5, 10)
range(5, 10)
>>> list(range(5, 10))
[5, 6, 7, 8, 9]
>>> list(range(10, 15))
[10, 11, 12, 13, 14]
>>> list(range(0, 10, 2))
[0, 2, 4, 6, 8]
>>>
```

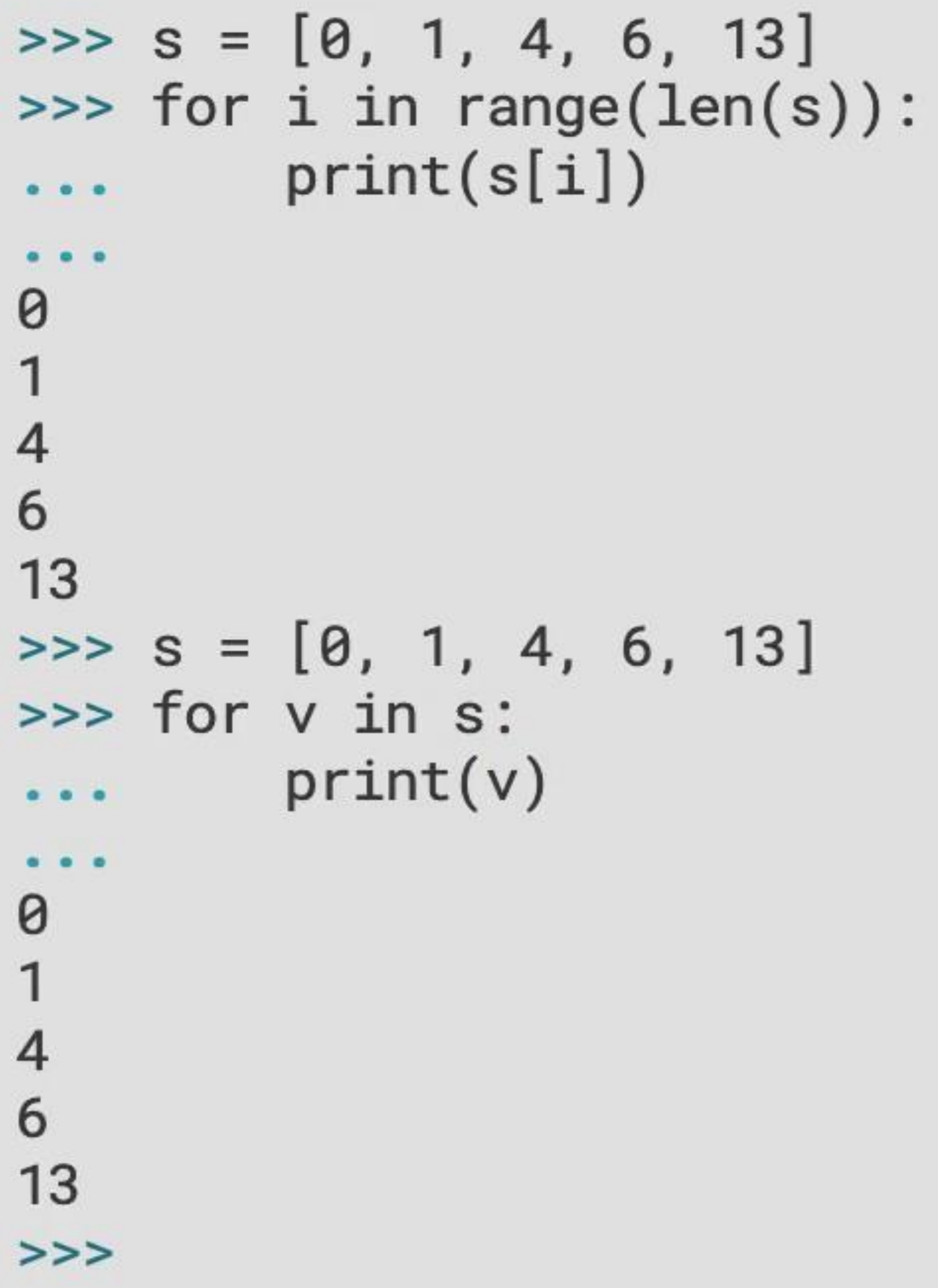
range() Signature

```
range(stop)

range(start, stop)

range(start, stop, step)
```

Range does not support keyword arguments



```
>>> s = [0, 1, 4, 6, 13]
>>> for i in range(len(s)):
...     print(s[i])
...
0
1
4
6
13
>>> s = [0, 1, 4, 6, 13]
>>> for v in s:
...     print(v)
...
0
1
4
6
13
>>>
```

Enumerate

enumerate

Constructs an iterable of `(index, value)` tuples around another iterable object

```
>>> t = [6, 372, 8862, 148800, 2096886]
>>> for p in enumerate(t):
...     print(p)
...
(0, 6)
(1, 372)
(2, 8862)
(3, 148800)
(4, 2096886)
>>> for i, v in enumerate(t):
...     print(f"i = {i}, v = {v}")
...
i = 0, v = 6
i = 1, v = 372
i = 2, v = 8862
i = 3, v = 148800
i = 4, v = 2096886
>>>
```