Python Built-in Functions

1. **abs()**

The `abs()` function returns the absolute value of a number.

Syntax:

```
abs(number)
```

Example:

```
print(abs(-5))  # Output: 5
```

2. **all()**

The `all()` function returns `True` if all elements in an iterable are true (or if the iterable is empty).

Syntax:

```
all(iterable)
```

Example:

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

3. **any()**

The `any()` function returns `True` if any element of an iterable is true. If the iterable is empty, it returns `False`.

Syntax:

```
any(iterable)
```

Example:

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

4. **bin()**

The `bin()` function returns the binary representation of an integer.

Syntax:

```
bin(number)
```

Example:

```
print(bin(10))  # Output: 0b1010
```

5. **bool()**

The `bool()` function returns a boolean value, `True` or `False`.

Syntax:

```
bool(value)
```

Example:

```
print(bool(0))  # Output: False
print(bool(1))  # Output: True
```

6. **float()**

The `float()` function returns a floating-point number from a number or a string.

Syntax:

```
float(value)
```

Example:

```
print(float(5))  # Output: 5.0
```

7. **int()**

The `int()` function returns an integer from a number or string.

Syntax:

```
int(value)
```

Example:

```
print(int(5.8))  # Output: 5
```

8. **len()**

The `len()` function returns the length of an object (like a list, string, tuple, etc.).

Syntax:

```
len(object)
```

Example:

```
print(len("Hello"))  # Output: 5
```

9. **max()**

The `max()` function returns the largest item in an iterable or the largest of two or more arguments.

Syntax:

```
max(iterable, *args)
```

Example:

```
print(max([1, 2, 3, 4]))  # Output: 4
```

10. **min()**

The `min()` function returns the smallest item in an iterable or the smallest of two or more arguments.

Syntax:

```
min(iterable, *args)
```

Example:

```
print(min([1, 2, 3, 4]))  # Output: 1
```

11. **pow()**

The `pow()` function returns the value of x to the power of y.

Syntax:

```
pow(x, y)
```

Example:

```
print(pow(2, 3))  # Output: 8
```

12. **round()**

The `round()` function rounds a number to a specified number of decimals.

Syntax:

```
round(number, digits)
```

Example:

```
print(round(5.5678, 2))  # Output: 5.57
```

13. **sum()**

The `sum()` function adds all the items in an iterable.

Syntax:

```
sum(iterable)
```

Example:

```
print(sum([1, 2, 3, 4]))  # Output: 10
```

14. **type()**

The `type()` function returns the type of an object.

Syntax:

```
type(object)
```

Example:

```
print(type(5))  # Output: <class 'int'>
```

Number Functions in Python

1. **ceil()**

The `ceil()` function returns the smallest integer greater than or equal to a given number. It is part of the `math` module.

Syntax:

```
import math
math.ceil(x)
```

Example:

```
import math
print(math.ceil(4.2))  # Output: 5
```

2. **floor()**

The `floor()` function returns the largest integer less than or equal to a given number.

Syntax:

```
import math
math.floor(x)
```

Example:

```
import math
print(math.floor(4.7))  # Output: 4
```

3. **sqrt()**

The `sqrt()` function returns the square root of a number.

Syntax:

```
import math
math.sqrt(x)
```

Example:

```
import math
print(math.sqrt(16))  # Output: 4.0
```

4. **log()**

The `log()` function returns the logarithm of a number.

Syntax:

```
import math
math.log(x)
```

Example:

```
import math
print(math.log(10))  # Output: 2.302585092994046
```

5. **factorial()**

The `factorial()` function returns the factorial of a number.

Syntax:

```
import math
math.factorial(x)
```

Example:

```
import math
print(math.factorial(5))  # Output: 120
```

6. **sin()**

The `sin()` function returns the sine of a number in radians.

Syntax:

```
import math
math.sin(x)
```

Example:

```
import math
print(math.sin(math.pi/2))  # Output: 1.0
```

7. **cos()**

The `cos()` function returns the cosine of a number in radians.

Syntax:

```
import math
math.cos(x)
```

Example:

```
import math
print(math.cos(0))  # Output: 1.0
```

8. **tan()**

The `tan()` function returns the tangent of a number in radians.

Syntax:

```
import math
math.tan(x)
```

Example:

```
import math
print(math.tan(math.pi/4))  # Output: 1.0
```

15. **filter()**

The `filter()` function constructs an iterator from elements of an iterable for which a function returns true.

Syntax:

```
filter(function, iterable)
```

Example:

```
def is_even(n):
    return n % 2 == 0

numbers = [1, 2, 3, 4, 5]
even_numbers = list(filter(is_even, numbers))
print(even_numbers)  # Output: [2, 4]
```

16. **map()**

The `map()` function applies a given function to all items in an iterable.

Syntax:

```
map(function, iterable)
```

Example:

```
def square(n):
    return n ** 2
```

```
numbers = [1, 2, 3, 4]
squared_numbers = list(map(square, numbers))
print(squared_numbers)  # Output: [1, 4, 9, 16]
```

17. **reduce()**

The `reduce()` function applies a rolling computation to sequential pairs of values in a list. It is part of the `functools` module.

Syntax:

```
from functools import reduce
reduce(function, iterable)
```

Example:

```
from functools import reduce

def add(x, y):
    return x + y

numbers = [1, 2, 3, 4]
sum_result = reduce(add, numbers)
print(sum_result)  # Output: 10
```

18. **zip()**

The `zip()` function returns an iterator of tuples, where the first tuple contains the first elements of each iterable, the second tuple contains the second elements, and so on.

Syntax:

```
zip(*iterables)
```

Example:

```
names = ['Alice', 'Bob', 'Charlie']
ages = [25, 30, 35]

zipped = list(zip(names, ages))
print(zipped)  # Output: [('Alice', 25), ('Bob', 30), ('Charlie', 35)]
```

19. **enumerate()**

The `enumerate()` function adds a counter to an iterable and returns it as an enumerate object.

Syntax:

```
enumerate(iterable, start=0)
```

Example:

```
fruits = ['apple', 'banana', 'cherry']
for index, fruit in enumerate(fruits):
    print(index, fruit)
# Output:
# 0 apple
# 1 banana
# 2 cherry
```

20. **sorted()**

The `sorted()` function returns a new sorted list from the elements of any iterable.

Syntax:

```
sorted(iterable, key=None, reverse=False)
```

Example:

```
numbers = [3, 1, 4, 1, 5]
sorted_numbers = sorted(numbers)
print(sorted_numbers)  # Output: [1, 1, 3, 4, 5]
```

21. **reversed()**

The `reversed()` function returns a reversed iterator.

Syntax:

```
reversed(seq)
```

Example:

```
numbers = [1, 2, 3, 4]
reversed_numbers = list(reversed(numbers))
```

```
print(reversed_numbers)  # Output: [4, 3, 2, 1]
```

22. **slice()**

The `slice()` function returns a slice object representing the set of indices specified by the range.

Syntax:

```
slice(start, stop, step)
```

Example:

```
my_list = [0, 1, 2, 3, 4, 5]
sliced_list = my_list[slice(1, 4)]
print(sliced_list)  # Output: [1, 2, 3]
```

23. **id()**

The `id()` function returns the identity of an object. This is an integer that is unique and constant for the object during its lifetime.

Syntax:

```
id(object)
```

Example:

```
x = 10
print(id(x))  # Output: (Some unique integer)
```

24. **input()**

The `input()` function allows user input from the console.

Syntax:

```
input(prompt)
```

Example:

```
name = input("Enter your name: ")
print("Hello, " + name)
```

25. **eval()**

The `eval()` function parses the expression passed to it and executes it as a valid Python expression.

Syntax:

```
eval(expression, globals=None, locals=None)
```

Example:

```
result = eval('3 + 5')
print(result)  # Output: 8
```

26. **exec()**

The `exec()` function executes the dynamically created program, which can either be a string or a code object.

Syntax:

```
exec(object, globals=None, locals=None)
```

Example:

```
code = "for i in range(5): print(i)"
exec(code)
# Output:
# 0
# 1
# 2
# 3
# 4
```