**Полный справочник встроенных функций Python с примерами**

Python предлагает богатый набор встроенных функций, которые доступны без необходимости импорта дополнительных модулей. Эти функции являются основой языка и значительно упрощают разработку. В этом подробном руководстве мы рассмотрим основных встроенных функций Python с практическими примерами и объяснениями.

Категории встроенных функций

Функции ввода-вывода и отображения

```
1. print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

Основная функция для вывода информации на экран. Поддерживает множество параметров для настройки форматирования:

```
print("Hello, world!")
print("Имя:", "Анна", "Возраст:", 25, sep=" | ")
print("Загрузка", end="...")
print("завершена!")
```

2. input(prompt="")

Получает строку от пользователя через стандартный ввод:

```
name = input("Введите ваше имя: ")
age = int(input("Введите ваш возраст: "))
print(f"Привет, {name}! Вам {age} лет.")
```

Совет: Всегда используйте преобразование типов для числовых данных, так как input() возвращает строку.

Функции для работы с последовательностями

3. len(s)

Возвращает количество элементов в объекте:

```
my_list = [1, 2, 3, 4, 5]
my_string = "Python"
my_dict = {"a": 1, "b": 2}

print(len(my_list))    # 5
print(len(my_string))  # 6
print(len(my_dict))    # 2
```

4. range(start, stop, step)

Создает последовательность чисел, часто используется в циклах:

```
# Различные варианты использования
for i in range(5):          # 0, 1, 2, 3, 4
    print(i)

for i in range(2, 8):       # 2, 3, 4, 5, 6, 7
    print(i)

for i in range(0, 10, 2):   # 0, 2, 4, 6, 8
    print(i)
```

5. enumerate(iterable, start=0)

Добавляет счетчик к итерируемому объекту:

```
fruits = ['яблоко', 'банан', 'вишня']
for index, fruit in enumerate(fruits, start=1):
    print(f"{index}. {fruit}")
# Выведет: 1. яблоко, 2. банан, 3. вишня
```

6. zip(*iterables)

Объединяет элементы нескольких итерируемых объектов:

```
names = ['Анна', 'Борис', 'Вера']
ages = [25, 30, 35]
cities = ['Москва', 'СПб', 'Казань']

for name, age, city in zip(names, ages, cities):
    print(f"{name}, {age} лет, живет в {city}")
```

Математические функции

7. abs(x)

Возвращает абсолютное значение числа (модуль):

```
print(abs(-7))      # 7
print(abs(3.14))    # 3.14
print(abs(-5+3j))   # 5.830951894845301 (для комплексных чисел)
```

8. sum(iterable, start=0)

Суммирует элементы итерируемого объекта:

```
numbers = [1, 2, 3, 4, 5]
print(sum(numbers))        # 15
print(sum(numbers, 10))    # 25 (начальное значение 10)
```

9. max(iterable) / max(arg1, arg2, *args)

Находит наибольший элемент:

```
print(max([1, 5, 3, 9, 2]))           # 9
print(max(1, 5, 3, 9, 2))             # 9
print(max(['apple', 'banana', 'cherry']))  # 'cherry' (лексикографически)
```

10. min(iterable) / min(arg1, arg2, *args)

Находит наименьший элемент:

```
print(min([1, 5, 3, 9, 2]))           # 1
print(min('hello'))                   # 'e' (минимальный символ)
```

11. round(number, ndigits=None)

Округляет число до указанного количества знаков:

```
print(round(3.14159))      # 3
print(round(3.14159, 2))   # 3.14
print(round(1234.5, -1))   # 1230.0
```

12. pow(base, exp, mod=None)

Возводит число в степень:

```
print(pow(2, 3))      # 8
print(pow(2, 3, 5))   # 3 (2^3 % 5)
print(pow(4, 0.5))    # 2.0 (квадратный корень)
```

13. divmod(a, b)

Возвращает частное и остаток от деления:

```
quotient, remainder = divmod(17, 5)
print(f"17 ÷ 5 = {quotient} остаток {remainder}")  # 17 ÷ 5 = 3 остаток 2
```

Функции преобразования типов

14. str(object)

Преобразует объект в строку:

```
print(str(123))        # '123'
print(str([1, 2, 3]))  # '[1, 2, 3]'
print(str(True))       # 'True'
```

15. int(x, base=10)

Преобразует в целое число:

```
print(int('123'))       # 123
print(int('1010', 2))   # 10 (двоичная система)
print(int('FF', 16))    # 255 (шестнадцатеричная)
print(int(3.14))        # 3
```

16. float(x)

Преобразует в число с плавающей точкой:

```
print(float('3.14'))    # 3.14
print(float('inf'))     # inf
print(float(5))         # 5.0
```

17. bool(x)

Преобразует в логическое значение:

```
print(bool(0))          # False
print(bool(1))          # True
print(bool(''))         # False
print(bool('hello'))    # True
print(bool([]))         # False
print(bool([1, 2]))     # True
```

18. complex(real, imag=0)

Создает комплексное число:

```
print(complex(3, 4))    # (3+4j)
print(complex('3+4j'))  # (3+4j)
print(complex(5))       # (5+0j)
```

Функции для работы с коллекциями

19. list(iterable)

Создает список из итерируемого объекта:

```
print(list('hello'))        # ['h', 'e', 'l', 'l', 'o']
print(list(range(5)))       # [0, 1, 2, 3, 4]
print(list((1, 2, 3)))      # [1, 2, 3]
```

20. tuple(iterable)

Создает кортеж из итерируемого объекта:

```
print(tuple([1, 2, 3]))     # (1, 2, 3)
print(tuple('hello'))       # ('h', 'e', 'l', 'l', 'o')
```

21. set(iterable)

Создает множество (уникальные элементы):

```
print(set([1, 2, 2, 3, 3]))  # {1, 2, 3}
print(set('hello'))          # {'h', 'e', 'l', 'o'}
```

22. frozenset(iterable)

Создает неизменяемое множество:

```
fs = frozenset([1, 2, 3])
print(fs)  # frozenset({1, 2, 3})
# fs.add(4)  # Ошибка! frozenset неизменяем
```

23. dict(mapping_or_iterable)

Создает словарь:

```
print(dict([('a', 1), ('b', 2)]))  # {'a': 1, 'b': 2}
print(dict(a=1, b=2))              # {'a': 1, 'b': 2}
print(dict(zip(['a', 'b'], [1, 2]))) # {'a': 1, 'b': 2}
```

Функции для работы с итераторами

24. iter(object, sentinel=None)

Создает итератор из объекта:

```
my_list = [1, 2, 3]
iterator = iter(my_list)
print(next(iterator))  # 1
print(next(iterator))  # 2
```

25. next(iterator, default=None)

Получает следующий элемент итератора:

```
numbers = iter([1, 2, 3])
print(next(numbers))        # 1
print(next(numbers))        # 2
print(next(numbers))        # 3
print(next(numbers, 'END')) # 'END' (значение по умолчанию)
```

26. reversed(sequence)

Создает обратный итератор:

```
print(list(reversed([1, 2, 3, 4])))  # [4, 3, 2, 1]
print(list(reversed('hello')))       # ['o', 'l', 'l', 'e', 'h']
```

27. sorted(iterable, key=None, reverse=False)

Возвращает отсортированный список:

```
print(sorted([3, 1, 4, 1, 5]))                    # [1, 1, 3, 4, 5]
print(sorted(['banana', 'apple', 'cherry']))      # ['apple', 'banana',
'cherry']
print(sorted([3, 1, 4, 1, 5], reverse=True))      # [5, 4, 3, 1, 1]
print(sorted(['apple', 'banana', 'cherry'], key=len))  # ['apple', 'banana',
'cherry']
```

Функции высшего порядка

28. map(function, iterable)

Применяет функцию к каждому элементу:

```
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))
print(squared)  # [1, 4, 9, 16, 25]

# Преобразование строк в числа
str_numbers = ['1', '2', '3', '4']
int_numbers = list(map(int, str_numbers))
print(int_numbers)  # [1, 2, 3, 4]
```

29. filter(function, iterable)

Фильтрует элементы по условию:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
print(even_numbers)  # [2, 4, 6, 8, 10]

# Фильтрация непустых строк
words = ['hello', '', 'world', '', 'python']
non_empty = list(filter(None, words))
print(non_empty)  # ['hello', 'world', 'python']
```

Логические функции

30. all(iterable)

Проверяет, все ли элементы истинны:

```
print(all([True, True, True]))    # True
print(all([True, False, True]))   # False
print(all([1, 2, 3]))             # True
print(all([1, 0, 3]))             # False
print(all([]))                    # True (пустая последовательность)
```

31. any(iterable)

Проверяет, есть ли хотя бы один истинный элемент:

```
print(any([False, False, False]))  # False
print(any([False, True, False]))   # True
print(any([0, 0, 1]))              # True
print(any([]))                     # False (пустая последовательность)
```

Функции для работы с объектами и атрибутами

32. type(object)

Возвращает тип объекта:

```
print(type(5))           # <class 'int'>
print(type('hello'))     # <class 'str'>
```

```
print(type([1, 2, 3]))   # <class 'list'>
print(type(lambda x: x)) # <class 'function'>
```

Функции для работы с памятью и идентификацией

40. id(object)

Возвращает уникальный идентификатор объекта:

```
a = [1, 2, 3]
b = [1, 2, 3]
c = a

print(id(a))     # Уникальный идентификатор
print(id(b))     # Другой идентификатор
print(id(c))     # Тот же, что и у 'a'
print(a is c)    # True
print(a is b)    # False
```

41. hash(object)

Возвращает хеш-значение объекта:

```
print(hash('hello'))    # Хеш строки
print(hash(42))         # Хеш числа
print(hash((1, 2, 3)))  # Хеш кортежа

# Списки не хешируемы
# print(hash([1, 2, 3]))  # Ошибка!
```

Функции для работы с символами и кодированием

43. ord(c)

Возвращает Unicode-код символа:

```
print(ord('A'))    # 65
print(ord('а'))    # 1072
print(ord('€'))    # 8364
```

44. chr(i)

Возвращает символ по Unicode-коду:

```
print(chr(65))     # 'A'
print(chr(1072))   # 'а'
print(chr(8364))   # '€'
```

45. ascii(object)

Возвращает ASCII-представление объекта:

```
print(ascii('hello'))      # "'hello'"
print(ascii('привет'))     # "'\u043f\u0440\u0438\u0432\u0435\u0442'"
print(ascii(['a', 'б']))   # "['a', '\u0431']"
```

Функции для работы с числовыми системами

46. bin(x)

Преобразует число в двоичную систему:

```
print(bin(10))    # '0b1010'
print(bin(255))   # '0b11111111'
print(bin(-5))    # '-0b101'
```

47. oct(x)

Преобразует число в восьмеричную систему:

```
print(oct(8))     # '0o10'
print(oct(64))    # '0o100'
print(oct(255))   # '0o377'
```

48. hex(x)

Преобразует число в шестнадцатеричную систему:

```
print(hex(255))   # '0xff'
print(hex(16))    # '0x10'
print(hex(1000))  # '0x3e8'
```

Функции для работы с файлами

49. open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)

Открывает файл для чтения или записи:

```
# Запись в файл
with open('example.txt', 'w', encoding='utf-8') as file:
    file.write('Привет, мир!')

# Чтение из файла
with open('example.txt', 'r', encoding='utf-8') as file:
    content = file.read()
    print(content)  # 'Привет, мир!'

# Чтение строк
with open('example.txt', 'r', encoding='utf-8') as file:
    for line in file:
        print(line.strip())
```

Функции для работы с байтами

50. bytes(source, encoding='utf-8', errors='strict')

Создает неизменяемый массив байтов:

```
print(bytes('hello', 'utf-8'))      # b'hello'
print(bytes([65, 66, 67]))          # b'ABC'
print(bytes(5))                     # b'\x00\x00\x00\x00\x00'
```

51. bytearray(source, encoding='utf-8', errors='strict')

Создает изменяемый массив байтов:

```
ba = bytearray('hello', 'utf-8')
print(ba)        # bytearray(b'hello')
ba[0] = 72       # Изменяем первый байт на 'H'
print(ba)        # bytearray(b'Hello')
```

52. memoryview(object)

Создает объект представления памяти:

```
data = bytearray(b'hello')
mv = memoryview(data)
print(mv[0])     # 104 (код символа 'h')
```

```
mv[0] = 72        # Изменяем на 'H'
print(data)       # bytearray(b'Hello')
```

Функции для работы с кодом

53. eval(expression, globals=None, locals=None)

Выполняет Python-выражение из строки:

```
      result = eval('2 + 3 * 4')
print(result)  # 14

x = 10
result = eval('x * 2 + 5')
print(result)  # 25

# Осторожно! eval может быть опасен
# Никогда не используйте с недоверенным вводом
```

54. exec(object, globals=None, locals=None)

Выполняет Python-код из строки:

```
      code = '''
def greet(name):
    return f"Привет, {name}!"

result = greet("Анна")
print(result)
'''

exec(code)  # Выведет: Привет, Анна!
```

55. compile(source, filename, mode, flags=0, dont_inherit=False, optimize=-1)

Компилирует исходный код в объект кода:

```
      code = compile('print("Hello, World!")', '<string>', 'exec')
exec(code)  # Выведет: Hello, World!

# Компиляция выражения
expr = compile('2 + 3', '<string>', 'eval')
result = eval(expr)
print(result)  # 5
```

Функции для работы с глобальными и локальными переменными

56. globals()

Возвращает словарь глобальных переменных:

```
      global_var = "Я глобальная"

def show_globals():
    print('global_var' in globals())  # True
    print(globals()['global_var'])    # "Я глобальная"

show_globals()
```

57. locals()

Возвращает словарь локальных переменных:

```
    def my_function():
    local_var = "Я локальная"
    param = 42
    print(locals())  # {'local_var': 'Я локальная', 'param': 42}

my_function()
```

Функции для работы с срезами

61. slice(start, stop, step=None)

Создает объект среза:

```
    my_list = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Создание срезов
s1 = slice(2, 8)       # Эквивалентно [2:8]
s2 = slice(1, 9, 2)    # Эквивалентно [1:9:2]

print(my_list[s1])     # [2, 3, 4, 5, 6, 7]
print(my_list[s2])     # [1, 3, 5, 7]
```

Функции для работы с форматированием

62. format(value, format_spec='')

Форматирует значение согласно спецификации:

```
    print(format(3.14159, '.2f'))        # '3.14'
print(format(1234, ','))             # '1,234'
print(format(255, 'x'))              # 'ff'
print(format(255, 'b'))              # '11111111'
print(format(0.5, '%'))              # '50.000000%'
```

63. repr(object)

Возвращает "официальное" строковое представление объекта:

```
    print(repr('hello'))        # "'hello'"
print(repr([1, 2, 3]))      # "[1, 2, 3]"
print(repr({'a': 1}))       # "{'a': 1}"

# Разница между str() и repr()
import datetime
now = datetime.datetime.now()
print(str(now))   # Читаемый формат
print(repr(now))  # Технический формат
```

Дополнительные функции

64. object()

Создает новый базовый объект:

```
    obj = object()
print(type(obj))    # <class 'object'>
print(dir(obj))     # Базовые методы объекта
```

65. help(object=None)

Выводит справочную информацию:

```
help(len)      # Справка по функции len
help(str)      # Справка по классу str
help(list.append)  # Справка по методу append
```

66. import(name, globals=None, locals=None, fromlist=(), level=0)

Динамически импортирует модуль:

```
# Осторожно! Обычно используйте обычный import
math_module = __import__('math')
print(math_module.sqrt(16))  # 4.0

# Лучше использовать importlib для динамического импорта
import importlib
json_module = importlib.import_module('json')
```

Ключевые моменты для запоминания:

- Встроенные функции доступны без импорта
- Они оптимизированы для производительности
- Многие функции универсальны и работают с разными типами данных