

Современные ВЕБ-технологии

02.03.03 -Математическое обеспечение и администрирование информационных систем,
направленность (профиль) -разработка и администрирование информационных систем

<http://vikchas.ru>

Лекция 2. Тема «Основы программирования»

Часовских Виктор Петрович
доктор технических наук, профессор кафедры
ШИиКМ, ФГБОУ ВО «Уральский государственный
экономический университет

Екатеринбург 2025

Основы программирования

1. Алгоритмические основы

Переменные и типы данных

Переменные — именованные области памяти, содержащие данные, которые могут изменяться.

Python:

```
age = 25          # Целое число (int)
price = 19.99     # Число с плавающей точкой (float)
name = "Алиса"    # Строка (str)
is_active = True  # Логическое значение (bool)
```

C#:

```
int age = 25;      // Целое число
double price = 19.99; // Число с плавающей точкой
string name = "Алиса"; // Строка
bool isActive = true; // Логическое значение
```

Операторы

Арифметические операторы

Python:

```
sum = a + b        # Сложение
diff = a - b       # Вычитание
product = a * b     # Умножение
quotient = a / b    # Деление
remainder = a % b   # Остаток от деления
power = a ** b      # Возведение в степень
```

C#:

```
int sum = a + b;    // Сложение
int diff = a - b;   // Вычитание
```

```
int product = a * b;    // Умножение
double quotient = a / b; // Деление
int remainder = a % b;  // Остаток от деления
double power = Math.Pow(a, b); // Возведение в степень
```

Операторы сравнения

Python:

```
is_equal = a == b    # Равно
not_equal = a != b    # Не равно
is_greater = a > b    # Больше
is_less = a < b       # Меньше
is_greater_eq = a >= b # Больше или равно
is_less_eq = a <= b   # Меньше или равно
```

C#:

```
bool isEqual = a == b;    // Равно
bool notEqual = a != b;   // Не равно
bool isGreater = a > b;   // Больше
bool isLess = a < b;      // Меньше
bool isGreaterEq = a >= b; // Больше или равно
bool isLessEq = a <= b;   // Меньше или равно
```

Логические операторы

Python:

```
result = a and b    # Логическое И
result = a or b      # Логическое ИЛИ
result = not a       # Логическое НЕ
```

C#:

```
bool result = a && b;   // Логическое И
bool result = a || b;   // Логическое ИЛИ
bool result = !a;       // Логическое НЕ
```

Выражения

Выражения — комбинация операторов и операндов, возвращающая значение.

Python:

Арифметическое выражение

```
area = (length + width) * 2
```

Логическое выражение

```
can_vote = age >= 18 and is_citizen
```

C#:

// Арифметическое выражение

```
double area = (length + width) * 2;
```

// Логическое выражение

```
bool canVote = age >= 18 && isCitizen;
```

2. Управляющие структуры

Условные конструкции

Python:

if-else

```
if age >= 18:
```

```
    print("Совершеннолетний")
```

```
else:
```

```
    print("Несовершеннолетний")
```

if-elif-else

```
if score >= 90:
```

```
    grade = "A"
```

```
elif score >= 80:
```

```
    grade = "B"
```

```
elif score >= 70:
```

```
    grade = "C"
else:
    grade = "D"
```

Тернарный оператор

```
status = "Взрослый" if age >= 18 else "Ребенок"
```

C#:

```
// if-else
```

```
if (age >= 18)
```

```
{
```

```
    Console.WriteLine("Совершеннолетний");
```

```
}
```

```
else
```

```
{
```

```
    Console.WriteLine("Несовершеннолетний");
```

```
}
```

```
// if-else if-else
```

```
if (score >= 90)
```

```
{
```

```
    grade = "A";
```

```
}
```

```
else if (score >= 80)
```

```
{
```

```
    grade = "B";
```

```
}
```

```
else if (score >= 70)
```

```
{
```

```
    grade = "C";
```

```
}  
else  
{  
    grade = "D";  
}
```

// Тернарный оператор

```
string status = age >= 18 ? "Взрослый" : "Ребенок";
```

Циклы

Python:

for цикл

```
for i in range(5):
```

```
    print(i) # Выводит числа от 0 до 4
```

while цикл

```
count = 0
```

```
while count < 5:
```

```
    print(count)
```

```
    count += 1
```

Перебор элементов коллекции

```
fruits = ["яблоко", "банан", "груша"]
```

```
for fruit in fruits:
```

```
    print(fruit)
```

break и continue

```
for i in range(10):
```

```
    if i == 3:
```

```
        continue # Пропустить итерацию
```

```
if i == 7:  
    break    # Выйти из цикла  
print(i)
```

C#:

```
// for цикл  
for (int i = 0; i < 5; i++)  
{  
    Console.WriteLine(i); // Выводит числа от 0 до 4  
}
```

```
// while цикл  
int count = 0;  
while (count < 5)  
{  
    Console.WriteLine(count);  
    count++;  
}
```

```
// Перебор элементов коллекции  
string[] fruits = {"яблоко", "банан", "груша"};  
foreach (string fruit in fruits)  
{  
    Console.WriteLine(fruit);  
}
```

```
// break и continue  
for (int i = 0; i < 10; i++)  
{  
    if (i == 3)
```

```
        continue; // Пропустить итерацию
    if (i == 7)
        break;    // Выйти из цикла
    Console.WriteLine(i);
}
```

Функции

Python:

Простая функция

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

Функция с параметрами по умолчанию

```
def power(base, exponent=2):
    return base ** exponent
```

Функция с произвольным числом аргументов

```
def sum_all(*numbers):
    return sum(numbers)
```

Вызов функций

```
message = greet("Иван")
square = power(4)      # 16
cube = power(3, 3)     # 27
total = sum_all(1, 2, 3, 4) # 10
```

C#:

// Простая функция

```
string Greet(string name)
{
    return $"Привет, {name}!";
}
```

```
}
```

```
// Функция с параметрами по умолчанию
```

```
double Power(double baseNum, int exponent = 2)
```

```
{
```

```
    return Math.Pow(baseNum, exponent);
```

```
}
```

```
// Функция с произвольным числом аргументов
```

```
int SumAll(params int[] numbers)
```

```
{
```

```
    int total = 0;
```

```
    foreach (int num in numbers)
```

```
        total += num;
```

```
    return total;
```

```
}
```

```
// Вызов функций
```

```
string message = Greet("Иван");
```

```
double square = Power(4);           // 16
```

```
double cube = Power(3, 3);          // 27
```

```
int total = SumAll(1, 2, 3, 4);     // 10
```

3. Структуры данных

Массивы

Python:

```
# В Python используются списки вместо классических массивов
```

```
numbers = [10, 20, 30, 40, 50]
```

```
print(numbers[0])    # 10 (первый элемент)
```

```
print(numbers[-1])    # 50 (последний элемент)
numbers[2] = 35       # Изменение элемента
length = len(numbers) # Длина массива
```

Срезы массивов

```
subset = numbers[1:4] # [20, 35, 40]
```

C#:

// Объявление и инициализация массива

```
int[] numbers = { 10, 20, 30, 40, 50 };
```

```
Console.WriteLine(numbers[0]);    // 10 (первый элемент)
```

```
numbers[2] = 35;                  // Изменение элемента
```

```
int length = numbers.Length;      // Длина массива
```

// Многомерный массив

```
int[,] matrix =
```

```
{
```

```
    {1, 2, 3},
```

```
    {4, 5, 6}
```

```
};
```

```
Console.WriteLine(matrix[1, 2]);  // 6
```

Списки

Python:

```
fruits = ["яблоко", "банан", "апельсин"]
```

```
fruits.append("груша")           # Добавление элемента в конец
```

```
fruits.insert(1, "манго")        # Вставка элемента
```

```
fruits.remove("банан")           # Удаление элемента
```

```
popped = fruits.pop()            # Удаляет и возвращает последний элемент
```

```
position = fruits.index("манго") # Поиск индекса элемента
```

```
is_present = "яблоко" in fruits # Проверка наличия элемента
```

Перебор списка с индексами

for i, fruit in enumerate(fruits):

print(f"{i}: {fruit}")

C#:

List<string> fruits = new List<string> {"яблоко", "банан", "апельсин"};

fruits.Add("груша"); // Добавление элемента в конец

fruits.Insert(1, "манго"); // Вставка элемента

fruits.Remove("банан"); // Удаление элемента

string popped = fruits[fruits.Count-1];

fruits.RemoveAt(fruits.Count-1); // Удаление последнего элемента

int position = fruits.IndexOf("манго"); // Поиск индекса элемента

bool isPresent = fruits.Contains("яблоко"); // Проверка наличия элемента

// Перебор списка с индексами

for (int i = 0; i < fruits.Count; i++)

{

Console.WriteLine(\$"{i}: {fruits[i]}");

}

Словари (хеш-таблицы)

Python:

person = {

"name": "Анна",

"age": 28,

"city": "Москва"

}

Доступ к элементам

print(person["name"]) # Анна

```
person["email"] = "anna@example.com" # Добавление элемента
person["age"] = 29                    # Изменение значения
```

```
# Безопасное получение значения
```

```
age = person.get("age", 0)    # Если ключа нет, возвращается 0
```

```
# Проверка наличия ключа
```

```
has_email = "email" in person # True
```

```
# Перебор ключей и значений
```

```
for key, value in person.items():
```

```
    print(f"{key}: {value}")
```

C#:

```
Dictionary<string, object> person = new Dictionary<string, object>
```

```
{
```

```
    {"name", "Анна"},
```

```
    {"age", 28},
```

```
    {"city", "Москва"}
```

```
};
```

```
// Доступ к элементам
```

```
Console.WriteLine(person["name"]);    // Анна
```

```
person["email"] = "anna@example.com"; // Добавление элемента
```

```
person["age"] = 29;                    // Изменение значения
```

```
// Безопасное получение значения
```

```
object age;
```

```
if (!person.TryGetValue("age", out age))
```

```
{
```

```
    age = 0; // Значение по умолчанию, если ключа нет  
}
```

```
// Проверка наличия ключа
```

```
bool hasEmail = person.ContainsKey("email"); // True
```

```
// Перебор ключей и значений
```

```
foreach (KeyValuePair<string, object> item in person)  
{  
    Console.WriteLine($"{item.Key}: {item.Value}");  
}
```

Деревья

Python:

```
class TreeNode:
```

```
    def __init__(self, value):  
        self.value = value  
        self.left = None  
        self.right = None
```

```
# Создание бинарного дерева
```

```
root = TreeNode(1)  
root.left = TreeNode(2)  
root.right = TreeNode(3)  
root.left.left = TreeNode(4)  
root.left.right = TreeNode(5)
```

```
# Обход дерева в глубину (DFS)
```

```
def inorder_traversal(node):  
    if node:
```

```
inorder_traversal(node.left)
print(node.value, end=' ')
inorder_traversal(node.right)
```

inorder_traversal(root) # Выведет: 4 2 5 1 3

C#:

```
class TreeNode
{
    public int Value;
    public TreeNode Left;
    public TreeNode Right;

    public TreeNode(int value)
    {
        Value = value;
        Left = null;
        Right = null;
    }
}

// Создание бинарного дерева
TreeNode root = new TreeNode(1);
root.Left = new TreeNode(2);
root.Right = new TreeNode(3);
root.Left.Left = new TreeNode(4);
root.Left.Right = new TreeNode(5);

// Обход дерева в глубину (DFS)
void InorderTraversal(TreeNode node)
```

```

{
    if (node != null)
    {
        InorderTraversal(node.Left);
        Console.Write(node.Value + " ");
        InorderTraversal(node.Right);
    }
}

```

InorderTraversal(root); // Выведет: 4 2 5 1 3

Графы

Python:

Представление графа с помощью списка смежности

```

graph = {
    'A': ['B', 'C'],
    'B': ['A', 'D', 'E'],
    'C': ['A', 'F'],
    'D': ['B'],
    'E': ['B', 'F'],
    'F': ['C', 'E']
}

```

Обход графа в ширину (BFS)

```

from collections import deque

```

```

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)

```

while queue:

vertex = queue.popleft()

print(vertex, end=' ')

for neighbor in graph[vertex]:

if neighbor not in visited:

visited.add(neighbor)

queue.append(neighbor)

bfs(graph, 'A') # Выведет: A B C D E F

C#:

using System;

using System.Collections.Generic;

// Представление графа с помощью списка смежности

Dictionary<char, List<char>> graph = new Dictionary<char, List<char>>

{

{'A', new List<char> {'B', 'C'}},

{'B', new List<char> {'A', 'D', 'E'}},

{'C', new List<char> {'A', 'F'}},

{'D', new List<char> {'B'}},

{'E', new List<char> {'B', 'F'}},

{'F', new List<char> {'C', 'E'}}

};

// Обход графа в ширину (BFS)

void BFS(Dictionary<char, List<char>> graph, char start)

{

```

HashSet<char> visited = new HashSet<char>();
Queue<char> queue = new Queue<char>();

visited.Add(start);
queue.Enqueue(start);

while (queue.Count > 0)
{
    char vertex = queue.Dequeue();
    Console.Write(vertex + " ");

    foreach (char neighbor in graph[vertex])
    {
        if (!visited.Contains(neighbor))
        {
            visited.Add(neighbor);
            queue.Enqueue(neighbor);
        }
    }
}

```

BFS(graph, 'A'); // Выведет: A B C D E F

Это основные концепции программирования с примерами на Python и C#. Они составляют фундамент, на котором строятся более сложные алгоритмы и программные системы.