

Лекция 9.

**Базовые элементы языков  
программирования.  
Подпрограммы**

# Подпрограммы

Подпрограммы – фундаментальная форма структурирования программы.

- Процедура
- Функция
- Метод

# Подпрограммы

Подпрограммы – фундаментальная форма структурирования программы.

- **Процедура** – независимая именованная часть программы, которую после однократного описания можно многократно вызвать по имени из последующих частей программы для выполнения определенных действий.
- **Функция** — подпрограмма специального вида, которая может возвращать результат. Вызов функции – это выражение языка программирования (может использоваться в других выражениях или в качестве правой части присваивания).
- **Метод** – подпрограмма, входящие в состав классов в объектных языках программирования.

# Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

## Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

- **Снижение сложности за счет формирования понятной промежуточной абстракции.** Выделение фрагмента кода в удачно названный метод — один из лучших способов документирования его цели.

# Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

- **Снижение сложности за счет формирования понятной промежуточной абстракции.** Выделение фрагмента кода в удачно названный метод — один из лучших способов документирования его цели.

```
Assign(f, "my.ini");
Open(f);
While (not eof(f)) do
  Begin
    Readln(f, s);
    If substr(s, 1,5) = "user" then UserName:=
substr(s, 6, len(s)-5);
  End;
```

## Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

- **Снижение сложности за счет формирования понятной промежуточной абстракции.** Выделение фрагмента кода в удачно названный метод — один из лучших способов документирования его цели.

```
Assign(f, "my.ini");
Open(f);
While (not eof(f)) do
  Begin
    Readln(f, s);
    If substr(s, 1,5) = "user" then UserName:=
substr(s, 6, len(s)-5);
  End;
```

```
UserName := GetUserFromIniFile;
```

## Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

- **Снижение сложности за счет формирования понятной промежуточной абстракции.**
- **Предотвращение дублирования кода.** Проще изменять код, выше надежность (меньше вероятность ошибиться).

## Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

- **Снижение сложности за счет формирования понятной промежуточной абстракции.**
- **Предотвращение дублирования кода.** Проще изменять код, выше надежность (меньше вероятность ошибиться).
- **Сокращение очередности действий.**

## Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

- **Снижение сложности за счет формирования понятной промежуточной абстракции.**
- **Предотвращение дублирования кода.** Проще изменять код, выше надежность (меньше вероятность ошибиться).
- **Сокращение очередности действий.**
- **Упрощение переноса приложения на другие платформы.**  
Выделение и изоляция неэкспортируемого кода (нестандартные возможности языка, зависимости от оборудования и операционной системы и т. д.).

## Подпрограммы

### Почему нужно писать подпрограммы и когда это нужно делать?

- **Снижение сложности за счет формирования понятной промежуточной абстракции.**
- **Предотвращение дублирования кода.** Проще изменять код, выше надежность (меньше вероятность ошибиться).
- **Соккрытие очередности действий.**
- **Упрощение переноса приложения на другие платформы.**
- **Упрощение сложных логических проверок.** Соккрытие деталей проверок + описательное имя метода позволяет лучше охарактеризовать суть проверки.

## Подпрограммы

Почему нужно писать подпрограммы и когда это нужно делать?

Причины создания методов = причины создания классов:

- Изоляция сложности и сокрытие деталей реализации.
- Ограничение влияния изменений.
- Соккрытие глобальных данных.
- Создание центральных точек управления.
- Облегчение повторного использования кода.

## Подпрограммы

Нужно ли создавать простые подпрограммы для простых целей?

Стоит ли писать подпрограммы из 2-3 строк?

# Подпрограммы

Пример (псевдокод):

```
points=deviceUnits*(POINTS_PER_INCH/DeviceUnitsPerInch())
```

Это выражение записано в нескольких местах в программе.

---

# Подпрограммы

Пример (псевдокод):

```
points=deviceUnits*(POINTS_PER_INCH/DeviceUnitsPerInch())
```

Это выражение записано в нескольких местах в программе.

---

```
Function DeviceUnitsToPoints ( deviceUnits Integer ):Integer
    DeviceUnitsToPoints = deviceUnits *
        (POINTS_PER_INCH/DeviceUnitsPerInch())
End Function
```

```
points = DeviceUnitsToPoints(deviceUnits)
```

# Подпрограммы

Пример (псевдокод):

```
points=deviceUnits*(POINTS_PER_INCH/DeviceUnitsPerInch())
```

Это выражение записано в нескольких местах в программе.

---

```
Function DeviceUnitsToPoints ( deviceUnits Integer ):Integer
    DeviceUnitsToPoints = deviceUnits *
        (POINTS_PER_INCH/DeviceUnitsPerInch())
End Function
```

```
points = DeviceUnitsToPoints(deviceUnits)
```

## Преимущества:

1. Легче читать код.
2. Проще изменять программу при усложнении задач.

## Связность подпрограмм

**Связность** – характеристика соответствия выполняемых в подпрограмме операций единой цели.

В идеале: подпрограмма эффективно решает одну задачу и больше ничего не делает.

В этом случае код будет надежным.

## Связность подпрограмм

**Связность** – характеристика соответствия выполняемых в подпрограмме операций единой цели.

В идеале: подпрограмма эффективно решает одну задачу и больше ничего не делает.

В этом случае код будет надежным.

Но как узнать, что считать "одной задачей"?

## Связность подпрограмм

Но как узнать, что считать "одной задачей"?

Одна задача может состоять из нескольких операций, но эти операции должны быть одного уровня абстракции – на один уровень ниже уровня этой задачи.

## Связность подпрограмм

Но как узнать, что считать "одной задачей"?

Одна задача может состоять из нескольких операций, но эти операции должны быть одного уровня абстракции – на один уровень ниже уровня этой задачи.

Подпрограмма **СкипятитьВоду**:

- Взять чайник
- Залить воду
- Поставить чайник на плиту
- Зажечь плиту
- Проверить, что вода закипела

# Связность подпрограмм

Правило последовательного понижения уровня абстракции.

## Подпрограмма **СкипятитьВоду**:

- Взять чайник
- Залить воду
  - Если водопровод, то из него.
  - Если нет водопровода, то из колодца.
- Поставить чайник на плиту
- Зажечь плиту
  - Если газовая, то спичкой.
  - Если электрическая, то кнопкой.
- Проверить, что вода закипела

## Связность подпрограмм

- **Функциональная связность.** Метод выполняет одну и только одну операцию. Это самый лучший вид связности.

*Sin()* – вычислить синус

*GetCustomerName()* – получить фамилию заказчика

*EraseFile()* – удалить файл

*CalculateLoanPayment()* – вычислить плату за кредит

*AgeFromBirthdate()* – определить возраст по дате рождения

## Связность подпрограмм

- **Функциональная связность.** Метод выполняет одну и только одну операцию.
- **Последовательная связность (sequential cohesion).** Метод содержит операции, которые обязательно выполняются в определенном порядке, используют данные предыдущих этапов и не формируют в целом единую функцию.

## Связность подпрограмм

- **Функциональная связность.** Метод выполняет одну и только одну операцию.
- **Последовательная связность (sequential cohesion).** Метод содержит операции, которые обязательно выполняются в определенном порядке, используют данные предыдущих этапов и не формируют в целом единую функцию.

Пример: метод вычисляет по дате рождения возраст сотрудника и по этому возрасту срок до его ухода на пенсию.

Как сделать этот метод функционально связанным?

## Связность подпрограмм

- **Функциональная связность.** Метод выполняет одну и только одну операцию.
- **Последовательная связность (sequential cohesion).** Метод содержит операции, которые обязательно выполняются в определенном порядке, используют данные предыдущих этапов и не формируют в целом единую функцию.

Пример: метод вычисляет по дате рождения возраст сотрудника и по этому возрасту срок до его ухода на пенсию.

Как сделать этот метод функционально связанным?

Создать два отдельных метода:

- метод, вычисляющий по дате рождения возраст сотрудника,
- метод, определяющий по дате рождения срок до ухода сотрудника на пенсию.

## Связность подпрограмм

- **Функциональная связность.** Метод выполняет одну и только одну операцию.
- **Последовательная связность.** Метод содержит операции, которые обязательно выполняются в определенном порядке, используют данные предыдущих этапов и не формируют в целом единую функцию.
- **Коммуникационная связность.** Выполняемые в методе операции используют одни и те же данные и не связаны между собой иным образом. Такие подпрограммы рекомендуется разделять на несколько.

## Связность подпрограмм

- **Функциональная связность.** Метод выполняет одну и только одну операцию.
- **Последовательная связность.** Метод содержит операции, которые обязательно выполняются в определенном порядке, используют данные предыдущих этапов и не формируют в целом единую функцию.
- **Коммуникационная связность.** Выполняемые в методе операции используют одни и те же данные и не связаны между собой иным образом. Такие подпрограммы рекомендуется разделять на несколько.
- **Временная связность.** Операции объединяются в метод на том основании, что все они выполняются в один интервал времени.
  - Startup()* – запуск программы
  - CompleteNewEmployee()* – прием нового сотрудника
  - Shutdown()* – завершение программы

## Связность подпрограмм

- **Функциональная связность.** Метод выполняет одну и только одну операцию.
- **Последовательная связность.** Метод содержит операции, которые обязательно выполняются в определенном порядке, используют данные предыдущих этапов и не формируют в целом единую функцию.
- **Коммуникационная связность.** Выполняемые в методе операции используют одни и те же данные и не связаны между собой иным образом. Такие подпрограммы рекомендуется разделять на несколько.
- **Временная связность.** Операции объединяются в метод на том основании, что все они выполняются в один интервал времени. В таких методах лучше не выполнять конкретных операций непосредственно, а вызывать для их выполнения другие методы.

## Связность подпрограмм

Остальные виды связности обычно неприемлемы (плохо организованный код).

- **Процедурная связность.** Операции в методе выполняются в определенном порядке.
- **Логическая связность.** Метод содержит разный функционал, а выбор выполняемой операции осуществляется на основе передаваемого в метод управляющего флага.

Пример: метод *InputAll()*, принимающий в зависимости от полученного флага фамилии клиентов, данные карт учета времени сотрудников или инвентаризационные данные.

Логическая связность оправдана, если единственная роль метода — координация выполнения команд и сам он не выполняет действий (обработчик событий).

## Имена подпрограмм

Как вы яхту назовете, так она и поплывет



## Имена подпрограмм

- **Описывайте все, что метод выполняет.**

```
Date newDate = date.add(5);
```

## Имена подпрограмм

- **Описывайте все, что метод выполняет.**

```
Date newDate = date.add(5);
```

Что прибавляется к дате (дни, часы, недели)?

Изменяется ли экземпляр date, или функция возвращает новое значение Date без изменения старого?

## Имена подпрограмм

- **Описывайте все, что метод выполняет.**

```
Date newDate = date.add(5);
```

Что прибавляется к дате (дни, часы, недели)?

Изменяется ли экземпляр `date`, или функция возвращает новое значение `Date` без изменения старого?

Если функция прибавляет пять дней с изменением `date`, то она должна называться `addDaysTo` или `increaseByDays`.

Если функция возвращает новую дату, смещенную на пять дней, но не изменяет исходного экземпляра `date`, то она должна называться `daysLater` или `daysSince`.

## Имена подпрограмм

- **Описывайте все, что метод выполняет.**
- **Избегайте невыразительных глаголов.**

*HandleCalculation(), PerformServices(), OutputUser(), ProcessInput()*

## Имена подпрограмм

- Описывайте все, что метод выполняет.
- Избегайте невыразительных глаголов.

*HandleCalculation(), PerformServices(), OutputUser(), ProcessInput()*  
*HandleOutput()* – непонятно. Лучше *FormatAndPrintOutput()*.

## Имена подпрограмм

- **Описывайте все, что метод выполняет.**
- **Избегайте невыразительных глаголов.**
- **Не используйте для различения имен методов исключительно номера.**

## Имена подпрограмм

- Описывайте все, что метод выполняет.
- Избегайте невыразительных глаголов.
- Не используйте для различения имен методов исключительно номера.
- Для именованя функции используйте описание возвращаемого значения.

*customerId.Next(), printer.IsReady() и pen.CurrentColor()*

## Имена подпрограмм

- Для именования процедуры используйте выразительный глагол, дополняя его объектом.

*PrintDocument(), CalcMonthlyRevenues(), CheckOrderInfo().*

В случае объектно-ориентированных языков имя объекта в имя процедуры включать не нужно: *document.Print(), orderInfo.Check().*

## Имена подпрограмм

- **Для именованя процедуры используйте выразительный глагол, дополняя его объектом.**
- **Аккуратно используйте антонимы.**
  - add/remove, increment/decrement, open/close, begin/end, insert/delete, show/hide
  - create/destroy, lock/unlock, source/target, first/last, min/max, start/stop
  - get/put, next/previous, up/down, get/set, old/new

Команда Get-Verb в PowerShell

## Имена подпрограмм

- Для именования процедуры используйте выразительный глагол, дополняя его объектом.
- Аккуратно используйте антонимы.
- Определяйте правила именования часто используемых операций и придерживайтесь их.

Метод, возвращающий уникальный идентификатор объекта:

```
employee.id.Get()  
dependent.GetId()  
supervisor()  
candidate.id()
```

## Имена подпрограмм

- Для именованной процедуры используйте выразительный глагол, дополняя его объектом.
- Аккуратно используйте антонимы.
- Определяйте правила именованной часто используемых операций и придерживайтесь их.
- Разделяйте команды и запросы.

Функция должна что-то делать или отвечать на какой-то вопрос, но не одновременно. Либо функция изменяет состояние объекта, либо возвращает информацию об этом объекте.

```
function Set(attribute: string, value: string): boolean;
```

```
...
```

```
if Set("username", "popov-av") ... // ???
```

## Имена подпрограмм

- Для именования процедуры используйте выразительный глагол, дополняя его объектом.
- Аккуратно используйте антонимы.
- Определяйте правила именования часто используемых операций и придерживайтесь их.
- **Разделяйте команды и запросы.**

Функция должна что-то делать или отвечать на какой-то вопрос, но не одновременно. Либо функция изменяет состояние объекта, либо возвращает информацию об этом объекте.

```
function Set(attribute: string, value: string): boolean;
```

```
...
```

```
if Set("username", "popov-av") ... // ???
```

**Лучше:**

```
if attributeExists("username") then setAttribute("username", "popov-av");
```

## Аргументы подпрограммы

Количество аргументов нужно минимизировать.

Идеальный (самый выразительный) случай – аргументов нет.  
Проще понимать и тестировать.

Следует избегать использования аргументов-флагов.

## Возврат из подпрограмм

Управляющие структуры для завершения работы подпрограммы  
return, exit

- **Используйте return, если это упростит подпрограмму**

```
Comparison Compare( int value1, int value2 ) {
    if ( value1 < value2 ) {
        return Comparison_LessThan;
    }
    else if ( value1 > value2 ) {
        return Comparison_GreaterThan;
    }
    return Comparison_Equal;
}
```

## Возврат из подпрограмм

Управляющие структуры для завершения работы подпрограммы `return`, `exit`

- **Используйте `return`**, если это упростит подпрограмму
- **Упрощайте сложную обработку ошибок с помощью досрочных `return`**

## Возврат из подпрограмм

Управляющие структуры для завершения работы подпрограммы  
`return, exit`

- **Используйте `return`, если это упростит подпрограмму**
- **Упрощайте сложную обработку ошибок с помощью досрочных `return`**

Неудачный код (Visual Basic):

```
If file.validName() Then
    If file.Open() Then
        If encryptionKey.valid() Then
            If file.Decrypt( encryptionKey ) Then
                ` Много кода.
            ...
        End If
    End If
End If
End If
```

## Возврат из подпрограмм

Управляющие структуры для завершения работы подпрограммы  
`return, exit`

- **Используйте `return`, если это упростит подпрограмму**
- **Упрощайте сложную обработку ошибок с помощью досрочных `return`**

Так лучше:

```
If Not file.validName() Then Exit Sub
If Not file.Open() Then Exit Sub
If Not encryptionKey.valid() Then Exit Sub
If Not file.Decrypt( encryptionKey ) Then Exit Sub
' Много кода.
...
```

## Возврат из подпрограмм

- Упрощайте сложную обработку ошибок с помощью досрочных `return`

Реалистичный код:

```
If Not file.validName() Then
    errorStatus = FileError_InvalidFileName
    Exit Sub
End If

If Not file.Open() Then
    errorStatus = FileError_CantOpenFile
    Exit Sub
End If

If Not encryptionKey.valid() Then
    errorStatus = FileError_InvalidEncryptionKey
    Exit Sub
End If

If Not file.Decrypt( encryptionKey ) Then
    errorStatus = FileError_CantDecryptFile
    Exit Sub
End If

` Много кода.
...
```