# Сопрограммы - поток управления

Пример: язык C# -

C# IEnumerator

yield return expression

yield break - завершение итераций

# Сопрограммы - поток управления

Пример: язык C# - async - await - асинхронные потоки управления

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.IO;

namespace HelloApp
{
    class Program
    {
        static async void ReadWriteAsync()
        {
            string s = "Hello world! One step at a time";

            // hello.txt - файл, который будет записываться и считываться
            using (StreamWriter writer = new StreamWriter("hello.txt", false))
            {
                await writer.WriteLineAsync(s);  // асинхронная запись в файл
            }
            using (StreamReader reader = new StreamReader("hello.txt"))
            {
                string result = await reader.ReadToEndAsync();  // асинхронное чтение из файла
                Console.WriteLine(result);
            }
        }
        static void Main(string[] args)
        {
            ReadWriteAsync();

            Console.WriteLine("Некоторая работа");

        }
    }
}
```

# Сопрограммы - поток управления

А откуда асинхронность?
class Task.....

```
namespace HelloApp
{
    class Program
    {
        static void Factorial()
        {
            int result = 1;
            for(int i = 1; i <= 6; i++)
            {
                result *= i;
            }
            Thread.Sleep(8000);
            Console.WriteLine($"Факториал равен {result}");
        }
        // определение асинхронного метода
        static async void FactorialAsync()
        {
            Console.WriteLine("Начало метода FactorialAsync"); // выполняется синхронно
            await Task.Run(Factorial);                         // выполняется асинхронно
            Console.WriteLine("Конец метода FactorialAsync");  // выполняется синхронно
        }

        static void Main(string[] args)
        {
            FactorialAsync();   // вызов асинхронного метода
            Console.WriteLine("Введите число: ");
            int n = Int32.Parse(Console.ReadLine());
            Console.WriteLine($"Квадрат числа равен {n * n}");
            Console.Read();
        }
    }
}
```

# Сопрограммы - поток управления

Генераторы и итераторы в Питоне

```
def fib(N):
    a = 1
    b = 1
    i = 0
    while i < N:
        yield a
        a, b = b, a + b
        i += 1
g = fib(10)
for i in g:
            print(i)
gg = fib(3)
next(gg); next(gg);next(gg);
next(gg)- ?  StopIteration
```

# Сопрограммы - поток управления

Сопрограммы в Питоне: PEP-342

генераторы - "почти" сопрограммы

Почему сопрограммы? - сохранение состояния

Почему "почти"? Нет возможности асинхронности (даже потенциальной)

В чем суть предложения - yield-выражение

(до этого был только yield-оператор)

yield expression # yield-оператор - в генераторе

Теперь в Питоне - только yield-выражения

data = (yield) # данные приходят при вызове: cor.send(d) - возвращает None

или data = (yield expression) # данные приходят при вызове: cor.send(d) - возвращает expression

Старый синтаксис (yield-оператор ) - yield-выражение с игнорируемым возвращенным значением

g.send(None) - эквивалентно next(g)

# Сопрограммы - поток управления

Сопрограммы в Питоне

```
def sum():
        count = 0
        while True:
                count += (yield) # yield-выражение
                print(count)
s = sum()
next(s) # можно s.send(None)
s.send(5) # 5 в качестве yield-выражения, напечатано 5
s.send(8) # напечатается 13
```

# Сопрограммы - поток управления

асинхронные сопрограммы в Питоне- async/await механизм (модули:  sys, asyncio ,  aiohttp   json)

```
loop = asyncio.get_event_loop()
client = aiohttp.ClientSession(loop=loop)
async def get_json(client, url):
    async with client.get(url) as response:
        assert response.status == 200
        return await response.read()
async def get_reddit_top(subreddit, client):
    data1 = await get_json(client, 'https://www.reddit.com/r/' + subreddit + '/top.json?sort=top&t=day&limit=5')
    j = json.loads(data1.decode('utf-8'))
    for i in j['data']['children']:
        score = i['data']['score']
        title = i['data']['title']
        link = i['data']['url']
        print(str(score) + ': ' + title + ' (' + link + ')')
    print('DONE:', subreddit + '\n')
asyncio.ensure_future(get_reddit_top('python', client))
asyncio.ensure_future(get_reddit_top('programming', client))
asyncio.ensure_future(get_reddit_top('compsci', client))
loop.run_forever()
```

Сильно менялся за последние 10 лет....

# Сопрограммы - поток управления

Горутины в ЯП Go

```
func foo(a int, b int) {
  fmt.Println(a,b);
}
foo(1,2)
foo(3,4)
go foo (1,2) // асинхронно
go foo (3,4) // асинхронно
```

Что делать с "гонками"?

Синхронизироваться.

Как?

"Легковесные" каналы:  var c chan T

Операции: (val - значение типа T)

- запись в канал c <- val
- чтение из канала  <- c (возвращает значение типа T)

# Сопрограммы - поток управления

Горутины в ЯП Go

"Легковесные" каналы:  var c chan T

Операции: (val - значение типа T)

- запись в канал c <- val
- чтение из канала  <- c (возвращает значение типа T)

По умолчанию - небуферизованные

c = make(chan T)

Буферизованные:

c = make(chan T, N) // N - емкость буфера

# Сопрограммы - поток управления

```
package main
import ( "fmt")
func fibonacci(n int, c chan int) {
        x, y := 0, 1
        for i := 0; i < n; i++ {
                c <- x
                fmt.Println("number...")
                x, y = y, x+y
        }
        close(c)
}
func main() {
        c := make(chan int)
        go fibonacci(10, c)
        for i := 0; i <10; i++ {
                x := <-c
                fmt.Println(x)
        }
}
```

# Сопрограммы - поток управления

```
package main

import ( "fmt"; "time")

func main() {
	tick := time.Tick(100 * time.Millisecond)
	boom := time.After(500 * time.Millisecond)
	for {
		select {
		case <-tick:
			fmt.Println("tick.")
		case <-boom:
			fmt.Println("BOOM!")
			return
		default:
			fmt.Println("    .")
			time.Sleep(50 * time.Millisecond)
		}
	}
}
```

# Подпрограммные типы данных

В "классическом" смысле - частный случай указательного типа - без явной операции разыменования.

Основные операции - вызов (), присваивание, == и !=

Область значений - имена (с сигнатурами) подпрограмм в проекте

(можно считать, что это - набор подпрограмм-констант, соответствующих сигнатуре)

C, Модула 2, Оберон, Ада 95.....

Ада 83 - подпрограммный тип вообще отсутствовал

А как они жили без него? Ответ - статическая параметризация

Ада 95 - появился по тем же причинам, что и указатели на любые переменные (access all'T).

# Подпрограммные типы данных

В "классическом" смысле - частный случай указательного типа - без явной операции разыменования.

TYPE PRC = PROCEDURE(VAR: INTEGER; REAL):REAL; -- Модула-2, Оберон

C++ - вариации на тему...

- указатели на глобальные функции
- указатели на статические функции-члены класса
- указатели на нестатические функции-члены класса

Ну и "до кучи":

- указатели на статические данные-члены класса
- указатели на нестатические данные-члены класса

Очень "тяжелый" и разнородный синтаксис

# Подпрограммные типы данных

C++ - вариации на тему...

```
class X {
        static double bar(int& a, double b);
        double prcX(int& a, double b);
};
X a;
```

- указатели на глобальные функции

```
typedef double (*PRC)(int&, double);
double foo(int& a, double b);
PRC pFn = foo;
```

- указатели на статические функции-члены класса

```
PRC pMem = X::bar; // то есть указатель на СФЧК === указатель на ГФ
```

- указатели на нестатические функции-члены класса

```
typedef double (X::*PRCX)(int&, double);
PRCX pFnX= &X::prcX;
int i;
double dd = (a.*pFnX)(i, 0.0);
```

# Подпрограммные типы данных

В "классическом" смысле - частный случай указательного типа - без явной операции разыменования.

Java (до 2014 года) - никаких вариаций - есть только методы, их нельзя делать параметрами, значениями переменных и т.д. (так как нет такого типа данных)

А как же они жили? Ответ - динамически связанные методы (виртуальные в терминах C++)

Пример: class Button со свойством Caption и реакцией на нажатие.

Если есть подпрограммный (функциональный) ТД => есть вариант - свойство OnClick() - ФТД

Если нет, то единственный вариант - иметь защищенный виртуальный метод Click(), который надо замещать в производном классе.

"Плодятся" однотипные классы => помощь - локальные классы (то есть классы, объявленные внутри блока кода). А блок кода - это тело метода какого-то класса.

Разновидность локального класса - анонимный класс

# Подпрограммные типы данных

Java - локальные и анонимные классы.

Критическая потребность для ЛокК и АнонК - доступ к переменным контекста (то есть они должны иметь доступ как к локальным переменным блока кода, так и к членам класса, в методе которого они объявлены).

Отсюда возникает общее понятие внутреннего класса (inner class), то есть класса, который объявлен внутри класса (внешнего), и имеет доступ к членам внешнего класса

Общее определение (для любого языка с классами): класс, объявленный внутри другого класса, называется вложенным классом.

# Подпрограммные типы данных

Вложенные классы - в C++, C# и большинстве других ЯП с классами - обычные классы, но с ограничением доступа  К НИМ (легко можно объявить как глобальные классы, но не хочется "засорять" глобальную ОВ). По реализации НИЧЕМ не отличаются от внешних (глобальных) классов.

Java - вложенные классы:

- статические
- внутренние
- локальные
- анонимные

# Подпрограммные типы данных

Внутренние ( + локальные + анонимные) классы ОЧЕНЬ близко подходят к концепции замыкания

Замыкание - это функция, которая использует в своем теле ссылки на свободные переменные (то есть переменные из контекста).

Нет своб. переменных => замыкание - просто функция.

Замыкание имеет смысл, только если в ЯП есть ФТД (то есть функции - это значения) - иначе говоря - функции первого порядка, а также локальные функции, которые могут использовать контекст.

Связанные термины - функции первого порядка, функции высшего порядка.

Замыкание = функция + "захваченный контекст"

Важный частный случай для локальных функций и замыканий - анонимные функции (лямбда-функции).

# ФТД и замыкания в ОИЯП

Java - главная проблема - отсутствие ФТД (в C++, C# уже были)

ЗАТО: идея замыкания уже реализована

Java 2014 (Java 8):

как реализовать понятие ФТД в языке, где нет понятия функции, а есть только методы?

"Language Hack"

Начиная с Java 2005: есть обобщенные классы и обобщенные интерфейсы

$\Rightarrow$в Java 8:

ФТД === интерфейс с единственным методом

Значения - анонимные функции и ссылки на методы класса