+ - есть
X - такая фича не поддерживается данным языком
- или <пусто> - еще нет примера кода.

Языки	C	C++	Python	Java	C#	Ada95
классы	X	+	+	+	+	+
функции		+	+	+	+	+
исключения	X		+	+/-	+	
циклы			+	+	+	+
рекурсия			+		+	
стат.полиморфизм			X	X	+	
дин. полиморфизм	X	+			+	
пар. полиморфизм	X	+	X	+	+	
итераторы	X		+	+		
абстрактный класс	X	+	X	+	+	
lambda	X		+		+	
функторы	X					
ввод/вывод			+	+	+	-/+

- C++
  - Проверка числа на простоту на этапе компиляции при помощи шаблонов
  - Пример с динамическим полиморфизмом и абстрактным классом
- C#
  - Пример использования базовых операций ввода/вывода, конструкций ветвления и циклов для вычисления наибольшего общего делителя, как рекурсивно, так и в цикле.
  - Пример использования наследования для реализации фигур: прямоугольника, круга, многоугольника и абстрактной функции, вычисляющей площадь.
  - Пример с перегрузкой операторов +, - и пользовательского преобразования типов
  - Пример определения интерфейса и статического полиморфизма с его методами
  - Пример параметрического полиморфизма
  - Использование лямбда-функций и делегатов
- Java
  - Merge sort. Здесь есть циклы, операторы ветвления, объявление классов, итераторы, стандартные контейнеры, параметрический полиморфизм
  - Параметрический полиморфизм + вывод в консоль
  - Пример реализации итерируемого класса
  - Пример абстрактного класса + наследование
- Python
  - Исключения
  - Пример с наследованием классов, циклами, вводом, выводом, списком
  - Лямбда-функция с рекурсией + ввод/вывод. (Факториал)
  - Пример функции-генератора (итераторы)
- Ada95
  - Пример с инкапсуляцией и наследованием
  - Классический GCD. Использование функций, циклов и вывода.

C++

Проверка числа на простоту на этапе компиляции при помощи шаблонов

```
struct false_type {
  typedef false_type type;
  enum { value = 0 };
};

struct true_type {
  typedef true_type type;
  enum { value = 1 };
};

template<bool condition, class T, class U>
struct if_ {
  typedef U type;
};

template <class T, class U>
struct if_<true, T, U> {
  typedef T type;
};

template<size_t N, size_t c>
struct is_prime_impl {
  typedef typename if_<(c*c > N),
                       true_type,
                       typename if_<(N % c == 0),
                          false_type,
                          is_prime_impl<N, c+1> >::type >::type type;
  enum { value = type::value };
};

template<size_t N>
struct is_prime {
  enum { value = is_prime_impl<N, 2>::type::value };
};
template <>
struct is_prime<0> {
  enum { value = 0 };
};
template <>
struct is_prime<1> {
  enum { value = 0 };
};
```

Пример с динамическим полиморфизмом и абстрактным классом

```
// Base class
class Shape {
public:
   // pure virtual function providing interface framework.
   virtual int getArea() = 0;
   void setWidth(int w) {
```

```
      width = w;
   }
   void setHeight(int h) {
      height = h;
   }
protected:
   int width;
   int height;
};

// Derived classes
class Rectangle: public Shape {
public:
   int getArea() {
      return (width * height);
   }
};

class Triangle: public Shape {
public:
   int getArea() {
      return (width * height)/2;
   }
};
```

C#

Пример использования базовых операций ввода/вывода, конструкций ветвления и циклов для вычисления наибольшего общего делителя, как рекурсивно, так и в цикле.

```
using System.IO;
using System;
class Program {
    static int GCD_rec(int a, int b) {
        if (b == 0)
            return a;
        else
            return GCD_rec(b, a % b);
    }
    static int GCD_rec2(int a, int b)
    { return b == 0 ? a : GCD_rec2(b, a % b); }
    static int GCD_loop(int a, int b) {
        while (b != 0) {
            int t = a;
            a = b;
            b = t % b;
        }
        return a;
    }
    static void Main() {
        Console.WriteLine("Enter the number: ");
        int testA = int.Parse(Console.ReadLine());
        Console.WriteLine("Enter the number: ");
        int testB = int.Parse(Console.ReadLine());
        Console.WriteLine("GCD_rec result: " + GCD_rec(testA, testB));
        Console.WriteLine("GCD_rec2 result: " + GCD_rec2(testA, testB));
```

```
        Console.WriteLine("GCD_loop result: " + GCD_loop(testA, testB));
    }
}
```

Пример использования наследования для реализации фигур: прямоугольника, круга, многоугольника и абстрактной функции, вычисляющей площадь.

```
using System.IO;
using System;
using System.Collections.Generic;
using System.Linq;

abstract class Shape {
    public abstract double Square();
}
class Rect : Shape {
    double w, h; // По умолчанию поле считается private
    public Rect(double w, double h) { this.w = w; this.h = h; }
    public override double Square()   { return w * h; }
}
class Circle : Shape {
    double radius;
    public Circle(double rad) { radius = rad; }
    public override double Square()    { return Math.PI * radius * radius; }
}
class Polygon : Shape {
    List<double> xs, ys;
    public Polygon(List<double> xs, List<double> ys) {
        if (xs.Count != ys.Count) {
            throw new Exception("Number of 'x' coordinates must match number of 'y'
coordinates");
        }
        this.xs = xs.ToList(); // ToList() вызывается для копирования списка
        this.ys = ys.ToList(); // иначе просто сохранилась бы ссылка
    }
    public override double Square() {
        double res = 0;
        for (int i = 0; i < xs.Count; i++) {
            int ni = (i + 1) % xs.Count;
            res += xs[ni] * ys[i] - xs[i] * ys[ni];
        }
        return Math.Abs(res) / 2;
    }
}

class Program {
    static void Main() {
        // Shape shap = new Shape(); <- ошибка, так нельзя, так как класс абстрактный
        Shape circ = new Circle(10);
        Shape rect = new Rect(5, 6);
        Console.WriteLine("circ square = " + circ.Square());
        Console.WriteLine("rect square = " + rect.Square());

        String input = Console.ReadLine();
        double[] nums = input
```

```
            .Split(' ') // разбиваем по пробелам
            .Where(s => !String.IsNullOrWhiteSpace(s))
            // оставляем только такие строки 's', которые не пустые
            .Select(s => double.Parse(s))
            // вместо каждой строки 's' выбираем число, которое она обозначает
            .ToArray();

        List<double>[] coords = new List<double>[2];
        coords[0] = new List<double>();
        coords[1] = new List<double>();

        for (int i = 0; i < nums.Length; i++) {
            coords[i % 2].Add(nums[i]);
        }

        try {
            Shape poly = new Polygon(coords[0], coords[1]);
            Console.WriteLine(poly.Square());
        } catch (Exception ex) {
            // если ввели нечётное количество чисел может произойти исключение
            Console.WriteLine(ex.Message);
        }
    }
}
```

Пример с перегрузкой операторов +, - и пользовательского преобразования типов

```
class Fraction {
    int num, den;
    public Fraction(int num, int den) {
        this.num = num;
        this.den = den;
    }
    public static Fraction operator +(Fraction a, Fraction b) {
        return new Fraction(a.num * b.den + b.num * a.den,
           a.den * b.den);
    }
    public static Fraction operator *(Fraction a, Fraction b) {
        return new Fraction(a.num * b.num, a.den * b.den);
    }
    // user-defined conversion from Fraction to double
    public static implicit operator double(Fraction f) {
        return (double)f.num / f.den;
    }
}
class Test {
    static void Main() {
        Fraction a = new Fraction(1, 2);
        Fraction b = new Fraction(3, 7);
        Fraction c = new Fraction(2, 3);
        Console.WriteLine((double)(a * b + c));
    }
}
```

Пример определения интерфейса и статического полиморфизма с его методами

```
interface ITest
{
   void F();                      // F()
   void F(int x);                 // F(int)
   void F(ref int x);             // F(ref int)
   void F(int x, int y);          // F(int, int)
   int F(string s);               // F(string)
   // int F(int x);               // F(int)  error повторное определение
   void F(string[] a);            // F(string[])
   // void F(params string[] a);  // F(string[]) error повторное определение
}
```

Пример параметрического полиморфизма

```
void PrintCountGeneric<T>(T collection) where T : ICollection
{
    Console.WriteLine(collection.Count);
}
...
var list = new List<int>() { 1, 2 };
PrintCountGeneric(list);
```

Использование лямбда-функций и делегатов

```
using System.IO;
using System;

class Program {
    delegate void Handler(int x);

    static void PrintX(int x) {
        Console.WriteLine(x);
    }
    static void PrintXX(int x) {
        Console.WriteLine(x * x);
    }
    static void Main() {
        // Func<int, int, double>
        // int, int - типы параметров
        // последний в списке тип double - тип возвращаемого значения
        Func<int, int, double> pow =
            (x, y) => Math.Pow(x, y);
        // Action<int, int>
        // int, int - типы параметров
        // возвращаемого значения в Action нет
        Action<int, int> printpow  =
            (x, y) => Console.WriteLine("pow: " + pow(x, y));

        printpow(2, 3);
        printpow(5, 2);
```

```
        Action<int> printpow4 = x => printpow(x, 4);

        // Handler - набор функций
        Handler h = PrintX;
        h += PrintXX;
        h += x => Console.WriteLine(x * x * x);
        h += new Handler(printpow4);
        Console.WriteLine("\nh(5):");
        h(5); // Тут все собранные функции вызовутся с параметром 5
    }
}
```

Java

Merge sort. Здесь есть циклы, операторы ветвления, объявление классов, итераторы, стандартные контейнеры, параметрический полиморфизм

```
import java.util.List;
import java.util.ArrayList;
import java.util.Iterator;

public class Merge{
    public static <E extends Comparable<? super E>> List<E> mergeSort(List<E> m){
        if(m.size() <= 1) return m;
        int middle = m.size() / 2;
        List<E> left = m.subList(0, middle);
        List<E> right = m.subList(middle, m.size());

        right = mergeSort(right);
        left = mergeSort(left);
        List<E> result = merge(left, right);
         return result;
    }

    public static <E extends Comparable<? super E>> List<E> merge(List<E> left, List<E>
right){
        List<E> result = new ArrayList<E>();
        Iterator<E> it1 = left.iterator();
        Iterator<E> it2 = right.iterator();

      E x = it1.next();
      E y = it2.next();
        while (true){

            if(x.compareTo(y) <= 0){
            result.add(x);
            if(it1.hasNext()){
                x = it1.next();
            }else{
                result.add(y);
                while(it2.hasNext()){
                  result.add(it2.next());
                }
                break;
            }
          }else{
```

```
            result.add(y);
            if(it2.hasNext()){
                y = it2.next();
            }else{
                result.add(x);
                while (it1.hasNext()){
                  result.add(it1.next());
                }
                break;
            }
          }
        }
        return result;
    }
}
```

Параметрический полиморфизм + вывод в консоль

```
public class Box<T> {

  private T t;
  public void add(T t) {
    this.t = t;
  }
  public T get() {
    return t;
  }

  public static void main(String[] args) {
     Box<Integer> integerBox = new Box<Integer>();
     Box<String> stringBox = new Box<String>();

     integerBox.add(new Integer(10));
     stringBox.add(new String("Hello World"));

     System.out.printf("Integer Value :%d\n\n", integerBox.get());
     System.out.printf("String Value :%s\n", stringBox.get());
  }
}
```

Пример реализации итерируемого класса

```
import java.util.NoSuchElementException;
import java.util.Iterator;
public class Range implements Iterable<Integer> {
    private int start, end;

    public Range(int start, int end) {
        this.start = start;
        this.end = end;
    }
    public Iterator<Integer> iterator() {
        return new RangeIterator();
    }
```

```
      // Inner class example
      private static final class RangeIterator implements
                              Iterator<Integer> {
            private int cursor;
            private final int end;

            public RangeIterator(int start, int end) {
                  this.cursor = start;
                  this.end = end;
            }
            public boolean hasNext() {
                  return this.cursor < end;
            }
            public Integer next() {
                  if(this.hasNext()) {
                        int current = cursor;
                        cursor ++;
                        return current;
                  }
                  throw new NoSuchElementException();
            }

            public void remove() {
                  throw new UnsupportedOperationException();
            }
      }

    public static void main(String[] args) {
        Range range = new Range(1, 10);

        // Long way
        Iterator<Integer> it = range.iterator();
        while(it.hasNext()) {
            int cur = it.next();
            System.out.println(cur);
        }
        // Shorter, nicer way:
        // Read ":" as "in"
        for(Integer cur : range) {
            System.out.println(cur);
        }
    }
}
```

Пример абстрактного класса + наследование

```
abstract class Instrument {
     protected String name;
     abstract public void play();
}

abstract class StringedInstrument extends Instrument {
     protected int numberOfStrings;
}

public class ElectricGuitar extends StringedInstrument {
     public ElectricGuitar() {
```

```
            super();
            this.name = "Guitar";
            this.numberOfStrings = 6;
      }
      public ElectricGuitar(int numberOfStrings) {
            super();
            this.name = "Guitar";
            this.numberOfStrings = numberOfStrings;
      }

      @Override
      public void play() {
            System.out.println("An electric " + numberOfStrings + "-string " + name
                        + " is rocking!");
      }
}

public class ElectricBassGuitar extends StringedInstrument {

      public ElectricBassGuitar() {
            super();
            this.name = "Bass Guitar";
            this.numberOfStrings = 4;
      }

      public ElectricBassGuitar(int numberOfStrings) {
            super();
            this.name = "Bass Guitar";
            this.numberOfStrings = numberOfStrings;
      }

      @Override
      public void play() {
            System.out.println("An electric " + numberOfStrings + "-string " + name
                        + " is rocking!");
      }
}
public class Execution {

      public static void main(String[] args) {
            ElectricGuitar guitar = new ElectricGuitar();
            ElectricBassGuitar bassGuitar = new ElectricBassGuitar();

            guitar.play();
            bassGuitar.play();

            guitar = new ElectricGuitar(7);
            bassGuitar = new ElectricBassGuitar(5);

            guitar.play();
            bassGuitar.play();
      }
}
```

Python

Исключения

```
try:
   raise ValueError #Так генерируем исключение
except ValueError:
   print('Поймали ошибку определенного вида')
except Exception:
   print('поймали любую ошибку')
else:
   print('Выводим это, если не поймали никакой ошибки')
finally:
   print('Выводим это независимо от того, была ошибка или нет ')
```

Пример с наследованием классов, циклами, вводом, выводом, списком

```
import math
class Figure(object):
    def __init__(self, center_x = 0, center_y = 0):
        self.x = center_x  # <--- self = this
        self.y = center_y
    def area(self):
        pass  # в python нет виртуальных методов

class Square(Figure):
    def __init__(self, size = 1):
        Figure.__init__(self)  # вызов родительского конструктора
        self.size = size
    def area(self):
        return self.size ** 2;

class Circle(Figure):
    def __init__(self, radius = 1):
        Figure.__init__(self)
        self.radius = radius
    def area(self):
        return math.pi * self.radius ** 2

if __name__ == "__main__":
    numbers = map(int, input('Enter numbers: ').split())
    circles = [Circle(x) for x in numbers]  # <--- list comprehension
    for c in circles:  # <--- цикл for
        print("Circle with r = {0} and area = {1}.".format(c.radius, c.area()))
```

Лямбда-функция с рекурсией + ввод/вывод. (Факториал)

```
factorial = lambda n : 1 if n == 0 else n * factorial(n - 1)

def factorial_simple(n):
    if n == 0:
        return 1
    else:
        return factorial_simple(n - 1) * n

a = int(input()) #input считывает строку
print(factorial(a)) #вызов функции
```

Пример функции-генератора (итераторы)

```
 def squares():
        n = 1
        while(True):
               yield n * n
               n += 1

 for x in squares():
        print(x)

 # output: 1 4 9 16 25 ...
```

Ada95

Пример с инкапсуляцией и наследованием

```
package File_System is
    type File is tagged private;
    procedure View(F : File);

    type Ada_File is new File with private;
    procedure View(F : Ada_File);

  private
    type File is tagged
     record
       Name : String(1..20);
     end record;

    type Ada_File is new File with
     record
       Compiled : Boolean := False;
     end record;
  end File_System;
```

Классический GCD. Использование функций, циклов и вывода.

```
with Ada.Text_Io; use Ada.Text_Io;

procedure Gcd_Test is
   function Gcd (A, B : Integer) return Integer is
      M : Integer := A;
      N : Integer := B;
      T : Integer;
   begin
      while N /= 0 loop
         T := M;
         M := N;
         N := T mod N;
      end loop;
      return M;
   end Gcd;

begin
   Put_Line("GCD of 100, 5 is" & Integer'Image(Gcd(100, 5)));
   Put_Line("GCD of 5, 100 is" & Integer'Image(Gcd(5, 100)));
   Put_Line("GCD of 7, 23 is" & Integer'Image(Gcd(7, 23)));
end Gcd_Test;
```