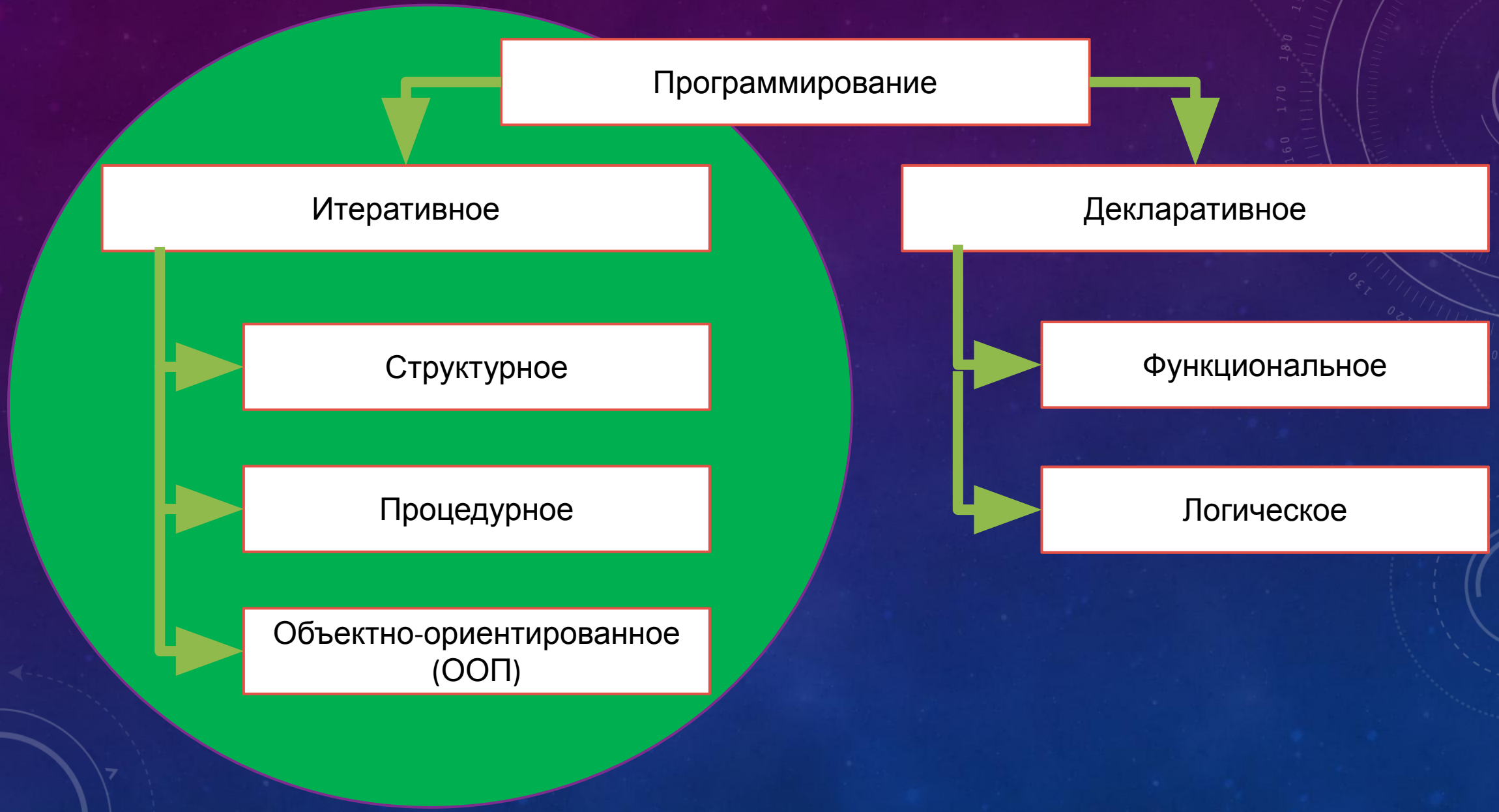
Парадигмы программирования

Хотя в любых случаев в программировании используются одни и те же базовые элементы (переменные, условия, циклы, массивы, функции), можно разными способами подходить к решению задач, объединяя эти простые элементы в более сложные(например в классы) и для разных задач одни способы подходят лучше других.

Т.Е это не какие-то серьезные различия в основах, а просто некоторые подходы и понятия для написания разных видов программ

Важно понять, что:

Современные языки высокого уровня позволяют использовать все или почти все способы, рассмотренные в дальнейшем (C#, python)


Программирование
Итеративное
Структурное
Процедурное
Объектно-ориентированное (ООП)
Декларативное
Функциональное
Логическое

Самая простая – Императивная (основа большинства других)

В этом подходе программы выглядят как простые последовательные операции, которые как-то взаимодействуют с данными. Т.Е. это просто какой-то код для решения несложной задачи. В нем используют только:

- Переменные
- Операторы(=, *, +, - и прочие более сложные)
- Переход на другие операции(go to)

```
int main()
{
    int x, y;
    cin >> x;
    y = 10 * x*x - 3 * x + 123;
    cout << y;
}
```

Используется в основном для написания программ решения не слишком сложных математических задач (например для решения полиномиальных уравнений, интегралов, любых других расчетов)

Почти все языки программирования могут использоваться для таких программ, и отличия будут совсем незначительные. Но в основном такой подход используется только в языках низкого уровня(ассемблер и некоторые другие)

Вторая – Структурная

Начиная примерно с 1970х годов стало понятно, что используя только простые операторы и “go to”, код быстро становится нечитаемым, особенно крупные программы.

Поэтому добавились такие элементы, как:

- Блоки кода(в c++ это {})
- Полностью заменены go to: вместо них теперь используются понятные циклы(for, while) и условия(if)

Собственно, почти все современные языки могут использоваться таким образом, но вот те которые задумывались именно для этой цели:

- C
- Pascal
- Basic

Третья – Процедурное

Начинают использоваться функции (процедуры), чтобы разделять выполнение программ, делить их на модули, использовать уже написанные функции снова, а не писать заново каждый раз

По сути такие идеи были предложены раньше, чем структурное программирование, поэтому почти во всех даже достаточно старых языках есть их поддержка, но первыми были:

- C
- Ada
- Фортран
- И многие другие..

Объектно-ориентированное (ООП)

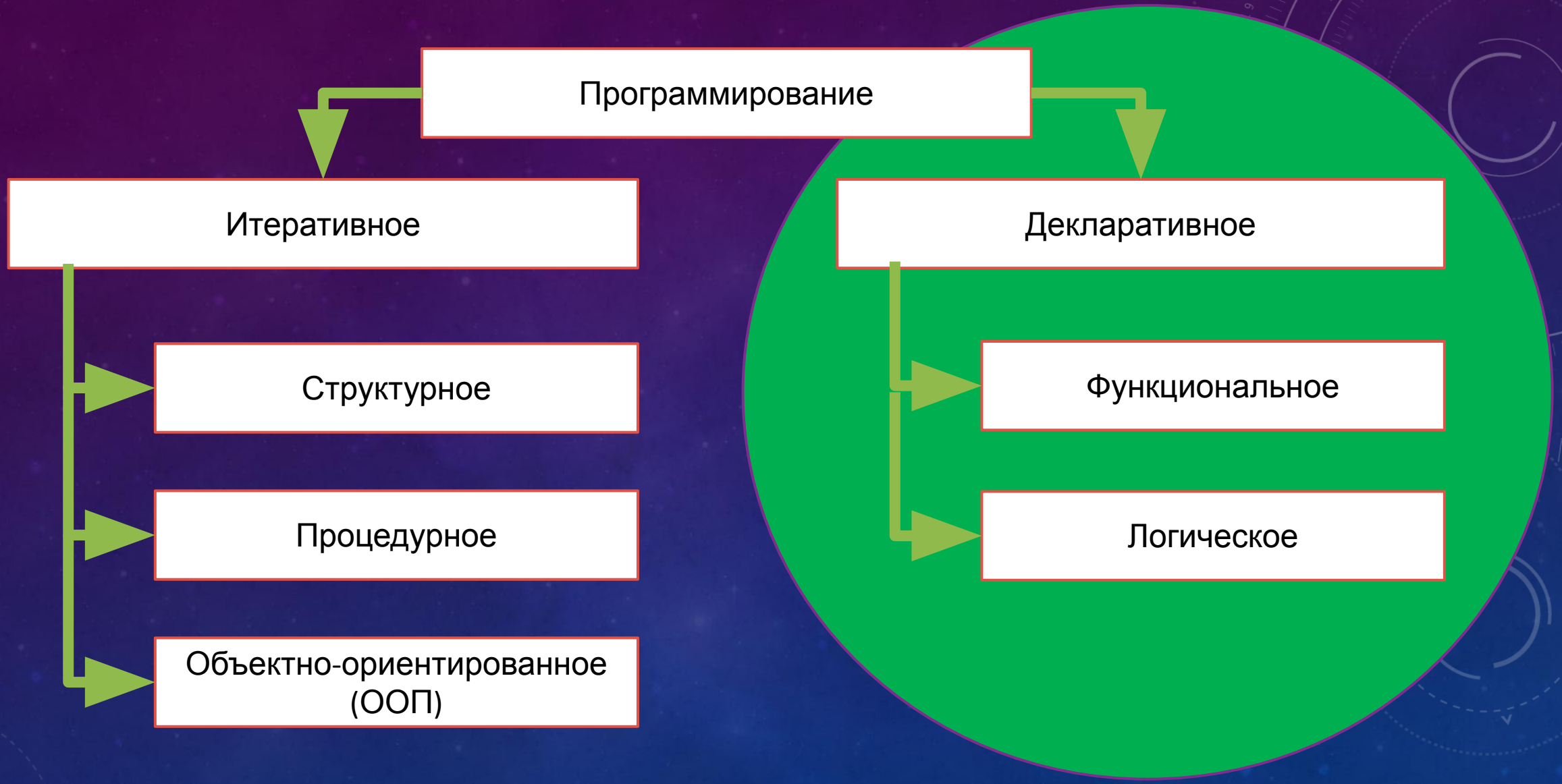
Опять встала та же самая проблема, что и раньше, что программы становились слишком большими, и нужно было придумать способы их разделить на отдельные части(теперь на классы)

Используется для написания сложных приложений из огромного количества элементов, а также для разделения разработки между несколькими людьми, где каждый делает классы для какой-то одной цели

- C#
- Java
- Python
- Delphi

```
class Animal {
public:
    int age;
    string animal_type;
    string name;
    string sound;

    void say() {
        cout << animal_type << ' ' << name << " говорит " << sound;
    }
};
```


Программирование
Итеративное
Структурное
Процедурное
Объектно-ориентированное (ООП)
Декларативное
Функциональное
Логическое

Декларативное программирование

Граница различий между итеративным и декларативным достаточно размыта, по сути определение такое: “Императивное программирование — это описание того, **как** ты делаешь что-то, а декларативное — того, **что** ты делаешь.” Но оно вообще почти ничего не дает понять про то, что происходит.

Одно из отличий, что в декларативном коде почти или совсем не будет операторов =, только вызовы различных функций, которые вызывают другие функции, выводят что-то и т.д.

Например, функция удвоения элементов массива:

Итеративный

```
function double (arr) {
  let results = []
  for (let i = 0; i < arr.length; i++){
    results.push(arr[i] * 2)
  }
  return results
}
```

Декларативный

```
function double (arr) {
  return arr.map((item) => item * 2)
}
```

То есть внутри конечно все эти функции и операции, которые мы используем, написаны итеративно, но нам не обязательно знать об их содержании, а только то, как они работают

И благодаря этому код становится намного более простым и понятным как для чтения, так и для написания

Еще пример: язык HTML, нам не нужно знать как браузер отрисовывает разные элементы, мы только определяем, что он будет рисовать

Часто этот способ используется в веб-разработке

```
1  <article>
2    <header>
3      <h1>Декларативное программирование</h1>
4      <p>Просто отрисуйся, я не знаю как</p>
5    </header>
6  </article>
```

Функциональное программирование

Используются различные функции с огромным количеством возможностей, почти не используя переменные и операторы

Самый яркий пример – Python, в нем очень много стандартных функциональных библиотек и библиотек для обработки данных, благодаря чему даже очень сложный код можно сжать до простых вызовов функций.

Императивный код

```
L = []
for x in xrange(10):
if x % 2 == 0:
if x**2>=50:
L.append(x)
else:
L.append(-x)
print L
```

Функциональный код

```
print [x**2>=50 and x or -x for x in xrange(10) if x%2==0]
```

Императивный Питон

```
data = [...]
sum = 0
for element in a:
sum += element ** 2
print sum
```

Функциональный Питон

```
data = [...]
sq = lambda x: x**2
sum = lambda x,y: x+y
print reduce(sum, map(sq, data))
```

Такой способ обычно комбинируют с итеративными, или пишут очень компактные скрипты для решения задач, анализа и обработки данных, машинного обучения и т.д.

- Haskell
- Python
- F#
- Lisp
- Wolfram