# Лекция #3. Выражения, функции, переменные и константы в Kotlin

## Математические функции в Kotlin

В стандартной библиотеке **Kotlin** есть широкий ассортимент функций, которые вы можете использовать в случае необходимости. Никогда неизвестно, когда может понадобиться использовать тригонометрию, особенно если вы серьезно работаете в **Kotlin** и программируете сложные игры, приложения либо нейронные сети.

На заметку: Не удаляйте код import kotlin.math.*, который поставляется с проектом, иначе **IntelliJ IDEA** сообщит, что не может найти данные функции.

Рассмотрим следующий пример:

```
1  sin(45 * PI / 180)
2  // 0.7071067811865475
3
4  cos(135 * PI / 180)
5  // -0.7071067811865475
```

Данный код вычисляет синус и косинус. Обратите внимание, что в обоих случаях используется число PI, которое является константой. **Kotlin** предоставляет нам число Пи с максимально возможной для компьютера точностью.

Рассмотрим также следующий пример:

```
1  sqrt(2.0)
2  // 1.414213562373095
```

Здесь вычисляется квадратный корень из числа 2.

Еще один пример:

```
max(5, 10)
// 10

min(-5, -10)
// -10
```

Если интересно, можете попробовать объединить все эти функции:

```
max(sqrt(2.0), PI / 2)
// 1.570796326794897
```

Обо всех математических функция можно прочитать в официальной документации к языку. Библиотека включают в себе следующие функции: тригонометрические, гиперболические, возведения в степень, логарифмические, округление и абсолютное значение.

## Переменные и константы в Kotlin

Программирование во многом строится на манипулировании данными. Помните, что все, отображаемое на экране, можно представить в виде чисел, которые отправляются в CPU. Хорошо, когда не возникает проблем при работе с простыми числовыми типами данных, но данные могут поступать и в более сложных формах вроде текста, изображений и коллекций.

В коде на **Kotlin** вы можете дать каждому фрагменту данных название, которое можно использовать, чтобы сослаться на него позже. Название ассоциируется с типом, указывая на вид данных вроде текста, чисел или даты.

### Константы в Kotlin

Взгляните на следующий пример:

```
val number: Int = 10
```

Здесь используется ключевое слово **val** для объявления константы под названием **number**, которая принадлежит к типу **Int**. Затем задается значение константы — 10.

На заметку: Вспомним операторы, здесь используется еще один. Знак равно **=** является оператором присваивания.

Тип **Int** может хранить целые числа. Десятичные числа можно хранить следующим образом:

```
1 | val pi: Double = 3.14159
```

Это похоже на константу типа **Int**, за тем исключением, что название и тип отличаются. На этот раз константа принадлежит к типу **Double**, который может хранить десятичные дроби с высокой точностью.

Существует также тип **Float**, который хранит десятичные дроби, но с меньшей точностью, чем **Double**. Точность **Double** примерно вдвое выше, чем у **Float**, поэтому он и называется **Double**. **Float** занимает меньше памяти, чем **Double**, но обычно использование памяти для чисел не является большой проблемой, поэтому **Double** используется в большинстве случаев.

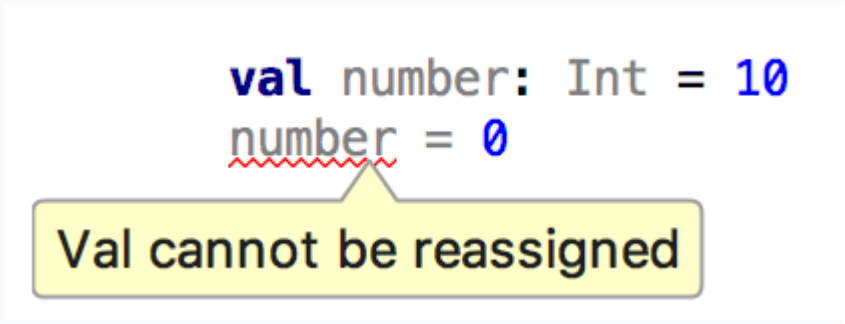
Несмотря на то, что мы называем элемент, созданный с помощью **val**, константой, правильнее будет сказать, что идентификатор, отмеченный ключевым словом **val**, является константой.

После объявления константы вы не сможете изменить ее. Рассмотрим следующий код:

```
1 | number = 0
```

Данный код выдает ошибку: **Val cannot be reassigned**

В **IDE** ошибка будет выглядеть следующим образом:




Константы полезны для значений, которые не изменятся. Например, если вы моделируете самолет и вам нужно отслеживать общее количество доступных мест, вы можете использовать константу.

Вы можете использовать константу для обозначения возраста человека. Несмотря на то, что возраст изменится по мере приближения дня рождения, вас может интересовать только их возраст в данный момент.

В определенных ситуациях, например, на верхнем уровне вашего кода вне каких-либо функций, вы можете добавить ключевое слово **const** к значению **val**, чтобы пометить его как константу во время компиляции:

```
1 const val reallyConstant: Int = 42
```

Значения, отмеченные ключевым словом **const**, должны быть инициализированы как **String** или как примитивный тип вроде **Int** или **Double**. Вы также можете использовать **const** внутри **Kotlin**-типа.

### Переменные в Kotlin

Часто вам может потребоваться изменить именованные данные. К примеру, при учете счета на банковском аккаунте будет лучше использовать переменную, а не константу.

Если бы данные программы никогда не менялись, эта программа была бы очень неудобной для применения в реальных условиях. Однако константу изменить нельзя.

Для данных, которые могут измениться, используют переменные, которые объявляются следующим образом:

```
1 var variableNumber: Int = 42
```

Отличается только первая часть выражения. При объявлении константы используется слово **val**, а при объявлении переменной — **var**.

После объявления переменной вы можете изменить ее на все, что угодно, но тип должен остаться прежним. К примеру, чтобы изменить переменную из примера выше, можно сделать следующее:

```
1 variableNumber = 0
2 variableNumber = 1_000_000
```

Для изменения переменной вы просто присваиваете ей новое значение.

На заметку: В **Kotlin** можно использовать нижнее подчеркивание, чтобы сделать длинные числа более читабельными. За количество и расположение подчеркиваний отвечаете вы сами.

## Правила именования переменных и констант в Kotlin

Всегда старайтесь выбирать понятные названия для переменных и констант. Хорошие названия выполняют роль документации и значительно облегчают процесс чтения кода. Они точно описывают роль переменной или константы. Далее дано несколько примеров хороших названий:

- personAge;

- numberOfPeople;
- gradePointAverage.

Зачастую **плохое название** является недостаточно понятным. Далее дано несколько примеров **плохих названий** для переменных:

- a;
- temp;
- average.

Главное , чтобы в будущем вам было понятно, к чему отсылается данная переменная или константа. Не полагайтесь на свою безупречную память. В программировании нередки случаи, когда вы оглядываетесь на код, написанный пару дней назад, и уже с трудом можете разобраться в нем. Упростите себе задачу, дав переменным и константам интуитивно понятные и точные названия.

Также обратите внимание на то, как написаны приведенные выше названия. В **Kotlin** принято использовать **CamelCase**. При именовании переменных и констант требуется соблюдать следующие правила:

- Название должно начинаться с маленькой буквы;
- При соединении нескольких слов в одно каждое слово, кроме первого, должно начинаться с заглавной буквы;
- Если какое-то из слов является аббревиатурой, все слово должно быть написано в одном регистре. Например, sourceURL или urlDescription.

## Инкремент и декремент значений в Kotlin

Одной из самых популярных операций является возможность уменьшения или увеличения значения переменной.

В **Kotlin** это можно сделать следующим образом:

```
var counter: Int = 0

counter += 1
// counter = 1

counter -= 1
// counter = 0
```

Сначала переменная **counter** равна 0. Операция инкремента изменяет ее значение на 1, и затем декремент возвращает ее значение к 0.

Данные операторы похожи на оператор присваивания **(=)**, только они также осуществляют сложение и вычитание. Они принимают текущее значение переменной, добавляют или вычитают данное значение и присваивают результат переменной.

Другими словами, код выше является сокращением для примера ниже:

```
var counter: Int = 0
counter = counter + 1
counter = counter - 1
```

Аналогично, операторы ***=** и **/=** выполняют операции умножения и деления:

```
counter = 10

counter *= 3  // same as counter = counter * 3
// counter = 30

counter /= 2  // same as counter = counter / 2
// counter = 15
```

## Как работает компьютер

Сперва разберем основы работы на компьютере, так как четкое понимание базы может значительно облегчить более сложные моменты программирования.

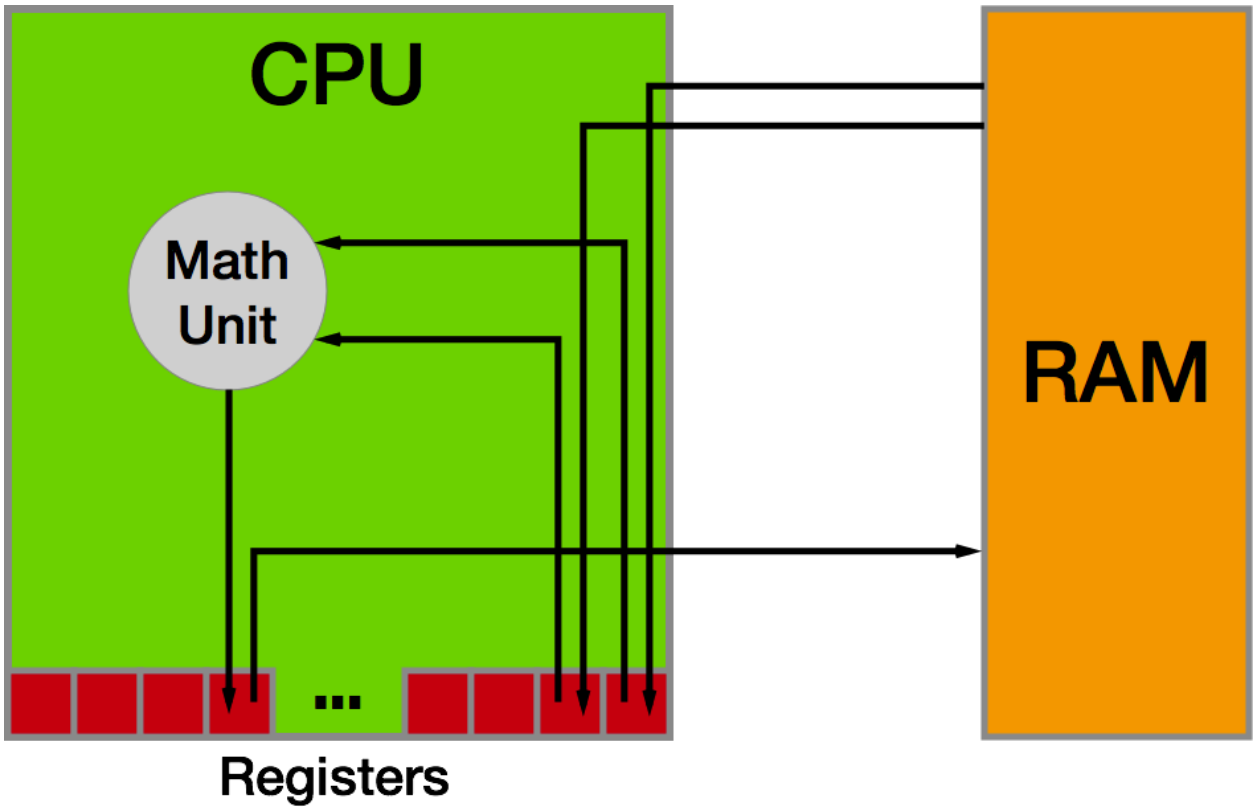
Может показаться странным, но на самом деле компьютер не очень умен. Сила компьютеров проистекает из того, как они программируются людьми. Если вы хотите успешно использовать возможности компьютера, важно понимать, как они работают.

Вы можете удивиться, узнав, что компьютеры сами по себе являются довольно простыми машинами. В основе компьютера лежит центральный процессор **(ЦП, CPU)**. По сути, это математическая машина. Он выполняет сложение, вычитание и другие арифметические операции с числами. Все, что вы видите при работе с компьютером, основано на процессоре, обрабатывающем числа миллионы раз в секунду. Разве не удивительно, что используются только цифры?

**CPU** сохраняет числа в небольших блоках памяти, которые называются регистрами. **CPU** может считывать числа в регистры из основной памяти компьютера, известной как оперативная память **(ОЗУ, RAM)**. Он также может записывать число, хранящееся в регистре, обратно в **RAM**. Это позволяет

процессору работать с большими объемами данных, которые не помещаются в банк регистров.

Далее дана схема того, как это работает:




Когда **CPU** извлекает значения из **RAM** в свои регистры, он использует эти значения в своей математической единице и сохраняет результаты обратно в другом регистре.

Каждый раз, когда **CPU** выполняет сложение, вычитание, чтение из **RAM** или запись в **RAM**, он выполняет одну инструкцию. Каждая компьютерная программа обычно состоит из тысяч или миллионов инструкций. Сложная компьютерная программа вроде операционной системы типа iOS, Android, macOS, Windows или Linux (да, это тоже «компьютерные программы») может содержать в общей сложности миллионы инструкций.

Вполне возможно написать отдельные инструкции, указывающие компьютеру, что делать, но для всех, кроме простейших программ, это было бы чрезвычайно трудоемко и утомительно. Это связано с тем, что большинство компьютерных программ нацелены на гораздо большее, чем простая математика — компьютерные программы позволяют вам путешествовать по Интернету, манипулировать изображениями и общаться с друзьями.

Вместо написания отдельных инструкций вы пишете код на определенном языке программирования, которым в нашем случае будет **Kotlin**. Этот код пропускается через компьютерную программу, называемую компилятором,

которая преобразует код в инструкции, которые **CPU** умеет выполнять. Каждая строка кода, который вы пишете, превратится во множество инструкций — некоторые строки могут оказаться десятками инструкций!

В случае **Kotlin**, который возник как язык для **Java Virtual Machine** или **JVM**, существует дополнительный уровень между компилятором и OS. Компилятор **Kotlin** создает так называемый байт-код, который запускается на **JVM** и попутно преобразуется в собственный код. **Kotlin** начал с **JVM**, но в настоящее время можно скомпилировать **Kotlin** непосредственно в собственный код.

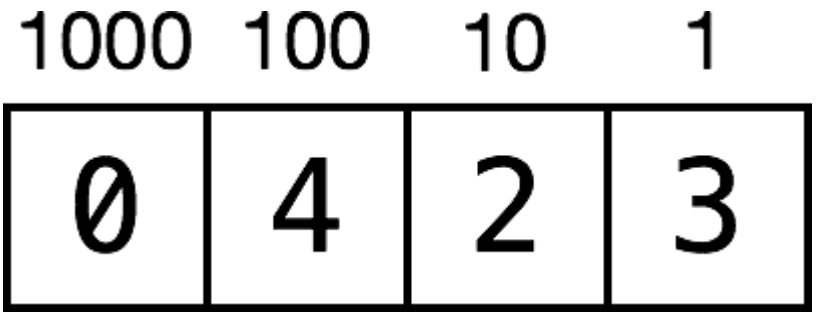
## Способы представления чисел в информатике

Вы наверняка знаете, что числа являются фундаментальной основой компьютера и всего того, что он делает. Любая информация, которую вы посылаете компилятору, в конечном итоге становится числом. К примеру, каждый символ внутри блока текста представлен числом.

Изображения не являются исключением. На компьютере каждая картинка представлена серией чисел. Изображение разделено на тысячи и даже миллионы элементов — пикселей, каждый из который является сплошным цветом. Если вы внимательно посмотрите на экран компьютера, вы сможете различить эти блоки. Однако это не получится сделать, если ваш дисплей с очень высоким разрешением, где пиксели невероятно малы. Каждый из этих сплошных цветных пикселей обычно представлен тремя числами — для красного, зеленого и синего **(RGB)**. Например, полностью красный пиксель будет на 100% красным, 0% зеленым и 0% синим.

Цифры, с которыми работает **CPU**, заметно отличаются от тех, к которым вы привыкли. В реальной жизни вы обычно работаете с десятичной системой. Используя эту систему счисления на протяжении длительного периода времени, вы интуитивно понимаете, как она работает. Чтобы вы могли понять точку зрения **CPU**, подумайте, как работает десятичная система.

Десятичное число 423 состоит из трех единиц, двух десятков и четырех сотен:




В десятичной системе каждая цифра числа может иметь значение 0, 1, 2, 3, 4, 5, 6, 7, 8 или 9, что дает в общей сложности 10 возможных значений для каждой цифры. Отсюда и название.

Однако истинное значение каждой цифры зависит от ее положения в числе. Двигаясь справа налево, каждая цифра умножается на возрастающую степень 10. Таким образом, множитель для крайней правой позиции равен 10 в степени 0, то есть 1. Двигаясь влево, следующий множитель будет равен 10 в степени. степень 1, то есть 10. Двигаясь снова влево, следующий множитель будет 10 в степени 2, которая равна 100. И так далее.

Это означает, что каждая цифра имеет значение в десять раз больше, чем цифра справа. Число 423 равно следующему:

```
1 (0 * 1000) + (4 * 100) + (2 * 10) + (3 * 1) = 423
```

## Бинарные, или двоичные числа

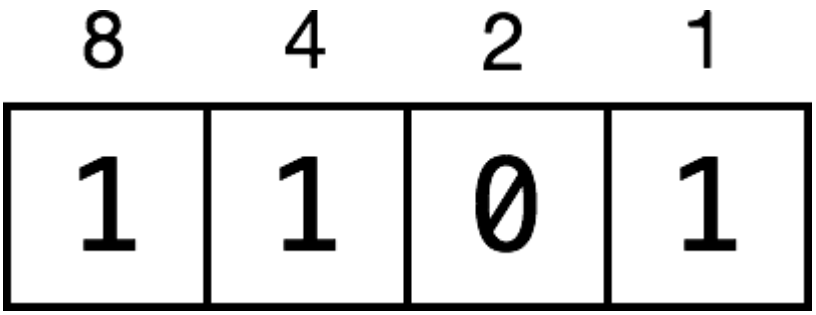
Так как вы постоянно оперируете десятичной системой, вы особо не задумываетесь над чтением чисел — процесс кажется естественным. Однако для компьютера десятичная система будет слишком сложной. Компьютеры работают с двоичной системой.

Вы наверняка слышали о бинарной, или двоичной системе. В ней есть только два варианта для цифры — 0 или 1.

Почти все современные компьютеры используют двоичный код, потому что на физическом уровне проще всего обрабатывать только два варианта для каждой цифры. В цифровой электронной схеме наличие электрического напряжения равно 1, а отсутствие — 0. Это двоичная система.

На заметку: Существовали как настоящие, так и воображаемые компьютеры, использующие троичную систему счисления, которая имеет три возможных значения вместо двух. Ученые-информатики, инженеры и преданные хакеры продолжают исследовать возможности компьютера с троичной системой.

Далее показано число 1101 в двоичной системе:




В десятичной системе значения разряда увеличиваются в 10 раз: 1, 10, 100, 1000 и так далее. В двоичной системе они увеличиваются в 2 раза: 1, 2, 4, 8, 16 и так

далее. Общее правило — умножать каждую цифру на возрастающую степень базового числа — в данном случае на степень двойки — двигаясь справа налево.

Таким образом, крайняя правая цифра представляет (1 * 2 ^ 0), то есть (1 * 1), то есть 1. Следующая цифра левее представляет (0 * 2 ^ 1), то есть (0 * 2), то есть 0. На приведенной выше схеме вы можете увидеть степень 2 над блоками.

Другими словами, каждая степень двойки либо равна (1), либо (0) не присутствует в качестве компонента двоичного числа. Десятичная версия двоичного числа — это сумма всех степеней двойки, составляющих это число. Итак, двоичное число 1101 равно:

```
(1 * 8) + (1 * 4) + (0 * 2) + (1 * 1) = 13
```

Если вам нужно конвертировать десятичное число 423 в двоичное, требуется просто разбить 423 на его компоненты со степенью 2. В результате вы получите следующее:

```
(1 * 256) + (1 * 128) + (0 * 64) + (1 * 32) + (0 * 16) + (0 * 8) + (1 * 4) + (1 * 2) + (1 * 1) = 423
```

Как видно в приведенном выше примере, полученное двоичное число будет 110100111. Вы можете доказать , что оно равно 423, выполнив математические вычисления. Компьютерным термином для каждой цифры двоичного числа является бит (**bit** сокращение от «**binary digit**», или «**бинарное число**»). Байт составляет восемь бит. Четыре бита — это полубайт, или ниббл.

**Ограниченная память компьютера** означает, что он обычно может обрабатывать числа до определенной длины. Каждый регистр обычно имеет длину 32 или 64 бита, поэтому мы говорим о 32-битных и 64-битных процессорах.

Следовательно, 32-разрядный процессор может обрабатывать максимальное базовое число 4 294 967 295, которое в двоичной системе равно 11111111111111111111111111111111. Это 32 единицы — посчитайте сами.

Компьютер может обрабатывать числа, превышающие максимум CPU, но вычисления должны разделяться и управляться.

## Шестнадцатеричные числа

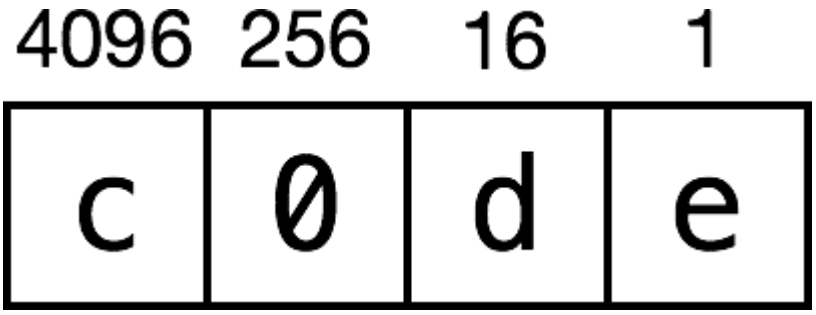
Работа с двоичными числами может стать утомительной, ведь на их запись уходит много времени. По этой причине в программировании часто используют шестнадцатеричную систему.

Ввиду отсутствия шестнадцати уникальных цифр, их только 10, используются первые шесть букв английского алфавита от a до f.

Эквиваленты десятичным числам в шестнадцатеричной системе:

- a = 10;
- b = 11;
- c = 12;
- d = 13;
- e = 14;
- f = 15.

Далее представлен пример шестнадцатеричного числа:




Обратите внимание, что шестнадцатеричные числа похожи на слова. Возможно, работать с ними будет веселее.

Теперь значение каждого числа соотносится в 16-ой степени. Как и раньше, вы можете конвертировать данное число в десятичное следующим образом:

```
(12 * 4096) + (0 * 256) + (13 * 16) + (14 * 1) = 49374
```

Можно перевести буквы в их десятичные эквиваленты и затем выполнить вычисления как обычно.

Но зачем?

Шестнадцатеричный формат важен, потому что каждая шестнадцатеричная цифра может представлять ровно четыре двоичных цифры. Двоичное число 1111 эквивалентно шестнадцатеричному f. Из этого следует, что вы можете просто объединить двоичные цифры, представляющие каждую шестнадцатеричную цифру, создавая шестнадцатеричное число, которое короче, чем его двоичные или десятичные эквиваленты.

Рассмотрим число c0de из примера выше:

```
c = 1100
0 = 0000
d = 1101
e = 1110

c0de = 1100 0000 1101 1110
```

Это оказывается довольно полезным, учитывая, как компьютеры используют длинные 32-битные или 64-битные двоичные числа. Напомним, что самое длинное 32-битное десятичное число — 4 294 967 295. В шестнадцатеричном формате это ffffffff. Это намного компактнее и понятнее.

## Как работает код

У компьютеров много ограничений, и сами по себе они могут делать немногое. Разработчики объединяют нужные элементы в правильном порядке для создания чего-то гораздо большего.

К примеру:

```
Step 1. Load photo from hard drive.
Step 2. Resize photo to 400 pixels wide by 300 pixels high.
Step 3. Apply sepia filter to photo.
Step 4. Print photo.
```

Такой код называют **псевдокодом**. В его основе не лежит никакой реально существующий язык программирования, код просто показывает алгоритм, которому нужно следовать в рассматриваемом случае. Согласно рассматриваемому здесь алгоритму, принимается фото, изменяется его размер, применяется фильтр и затем оно выводится.

Код на **Kotlin** точно такой же: это пошаговый список инструкций для компьютера. С каждым новым уроком данные инструкции будут становиться все сложнее, но принцип всегда тот же: вы просто сообщаете компьютеру, что нужно делать, шаг за шагом.

Каждый язык программирования представляет собой предопределенный высокоуровневый способ выражения данных шагов. Компилятор знает, как интерпретировать код, который вы пишете, и преобразовывать его в инструкции, которые может выполнять **CPU**.

Существует много разных языков программирования, каждый со своими достоинствами и недостатками. На сегодняшний день **Kotlin** является одним из самых современных, объединяя как сильные, так и слабые стороны языков прошлого. Это развивающийся язык с огромным потенциалом.