2021

**Transport and Telecommunication Institute**



**Основы языка Python,**

**базовое программирование NAO на Python.**

Николай Котцов

[**1.** **Основы Python** 3](#_Toc71905761)

[**1.1.** **Особенности языка** 3](#_Toc71905762)

[**1.2.** **Как писать программы на Python** 3](#_Toc71905763)

[**1.3.** **Создание скриптов** 3](#_Toc71905764)

[**1.4.** **Данные и их типы** 3](#_Toc71905765)

[**1.5.** **Операции в программировании** 4](#_Toc71905766)

[**1.6.** **Вывод данных. Функция print()** 4](#_Toc71905767)

[**1.7.** **Логические выражения** 5](#_Toc71905768)

[**1.8.** **Ветвление. Условный оператор** 6](#_Toc71905769)

[**1.9.** **Циклы в программировании** 7](#_Toc71905770)

[**Цикл while** 7](#_Toc71905771)

[**Цикл for** 7](#_Toc71905772)

[**1.1.** **Функции в программировании** 8](#_Toc71905773)

[**1.2.** **Локальные и глобальные переменные** 8](#_Toc71905774)

[**1.3.** **Возврат значений из функции. Оператор return** 9](#_Toc71905775)

[**1.4.** **Параметры и аргументы функции** 9](#_Toc71905776)

[**1.5.** **Модули** 10](#_Toc71905777)

[**1.6.** **Списки** 10](#_Toc71905778)

[**1.7.** **Функции первого класса и функция высшего порядка** 11](#_Toc71905779)

[**2.** **Базовое программирование NAO на Python** 12](#_Toc71905780)

[**2.1.** **Задание 1** 12](#_Toc71905781)

[**1.1.1.** **Задание 1.1** 15](#_Toc71905782)

[**1.1.2.** **Задание 1.2** 17](#_Toc71905783)

[**1.2.** **Задание 2** 18](#_Toc71905784)

[**1.2.1.** **Задание 2.1** 19](#_Toc71905785)

[**1.3.** **Задание 3** 21](#_Toc71905786)

[**1.4.** **Задание 4** 23](#_Toc71905787)

[**1.4.1.** **Задание 4.1** 27](#_Toc71905788)

[**1.4.2.** **Задание 4.2** 31](#_Toc71905789)

[**3.** **Дополнения** 33](#_Toc71905790)

[**3.1.** **Контактные и тактильные сенсоры** 33](#_Toc71905791)

[**3.1.1.** **Сенсоры головы** 33](#_Toc71905792)

[**3.1.2.** **Сенсоры рук** 34](#_Toc71905793)

[**3.1.3.** **Кнопка на груди** 34](#_Toc71905794)

[**3.1.4.** **Сенсоры ног** 35](#_Toc71905795)

[**3.2.** **Таблица светодиодов** 36](#_Toc71905796)

[**3.2.1.** **Тактильные сенсоры головы** 36](#_Toc71905797)

[**3.2.2.** **Глаза** 38](#_Toc71905798)

[**3.2.3.** **Уши** 40](#_Toc71905799)

[**3.2.4.** **Ноги** 42](#_Toc71905800)

[**3.2.5.** **Кнопка на груди** 43](#_Toc71905801)

[**3.2.6.** **Дополнительно** 43](#_Toc71905802)

1. **Основы Python**
   1. **Особенности языка**

Python – интерпретируемый язык программирования. Это значит, что исходный код частями преобразуется в машинный в процессе его чтения специальной программой – интерпретатором.

Python характеризуется ясным синтаксисом. Читать код на нем легче, чем на других языках программирования, так как в Питоне мало используются такие вспомогательные синтаксические элементы как скобки, точки с запятыми. С другой стороны, правила языка заставляют программистов делать отступы для обозначения вложенных конструкций. Понятно, что хорошо оформленный текст с малым количеством отвлекающих элементов читать и понимать легче.

Python – это полноценный во многом универсальный язык программирования, используемый в различных сферах. Основная, но не единственная, поддерживаемая им парадигма, – объектно-ориентированное программирование. Однако будем изучать структурное программирование, так как оно является базой. Без знания основных типов данных, ветвлений, циклов, функций нет смысла изучать более сложные парадигмы, так как в них все это используется.

* 1. **Как писать программы на Python**

Грубо говоря, интерпретатор выполняет команды построчно. Пишешь строку, нажимаешь **Enter**, интерпретатор выполняет ее, наблюдаешь результат.

Это удобно, когда изучаешь особенности языка или тестируешь какую-нибудь небольшую часть кода. Ведь если работать на компилируемом языке, пришлось бы сначала создать файл с кодом на исходном языке программирования, затем передать его компилятору, получить от него исполняемый файл и только потом выполнить программу и оценить результат. К счастью, даже в случае с компилируемыми языками все эти действия выполняет среда разработки, что упрощает жизнь программиста.

* 1. **Создание скриптов**

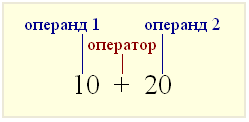
Несмотря на удобства интерактивного режима, чаще всего необходимо сохранить исходный программный код для последующего выполнения и использования. В таком случае подготавливаются файлы, которые передаются затем интерпретатору на исполнение. Файлы с кодом на Python обычно имеют расширение **.py**.

* 1. **Данные и их типы**

При знакомстве с языком программирования Python мы столкнемся с тремя типами данных:

* **целые числа** (тип **int**) – положительные и отрицательные целые числа, а также 0 (например, 4, 687, -45, 0).
* **числа с плавающей точкой** (тип **float**) – дробные, они же вещественные, числа (например, 1.45, -3.789654, 0.00453). Примечание: для разделения целой и дробной частей здесь используется точка, а не запятая.
* **строки** (тип **str**) — набор символов, заключенных в кавычки (например, "ball", "What is your name?", 'dkfjUUv', '6589'). Примечание: кавычки в Python могут быть одинарными или двойными; одиночный символ в кавычках также является строкой, отдельного символьного типа в Питоне нет.
  1. **Операции в программировании**

**Операция** – это выполнение каких-либо действий над данными, которые в данном случае именуют **операндами**. Само действие выполняет **оператор** – специальный инструмент. Если бы вы выполняли операцию постройки стола, то вашими операндами были бы доска и гвоздь, а оператором – молоток.



Так в математике и программировании символ плюса является оператором операции сложения по отношению к числам. В случае строк этот же оператор выполняет операцию конкатенации, то есть соединения.

>>> 10.25 + 98.36

108.61

>>> 'Hello' + 'World'

'HelloWorld'

Здесь следует для себя отметить, что то, что делает оператор в операции, зависит не только от него, но и от типов данных, которыми он оперирует. Молоток в случае нападения на вас крокодила перестанет играть роль строительного инструмента. Однако в большинстве случаев операторы не универсальны. Например, знак плюса неприменим, если операндами являются, с одной стороны, число, а с другой – строка.

>>> 1 + 'a'

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for +:

'int' and 'str'

Здесь в строке TypeError: unsupported operand type(s) for +: 'int' and 'str' интерпретатор сообщает, что произошла ошибка типа – неподдерживаемый операнд для типов **int** и **str**.

* 1. **Вывод данных. Функция print()**

Что такое функция в программировании, узнаем позже. Пока будем считать, что **print()** – это такая команда языка Python, которая выводит то, что в ее скобках на экран. В скобках могут быть любые типы данных. Кроме того, количество данных может быть различным:

>>> print(2.34)

2.34

>>> print("Hello")

Hello

>>> print("a:", 1)

a: 1

>>> print(10 - 2.5/2)

8.75

Следующее, что стоит рассказать о функции **print** – это использование форматирования строк. На самом деле оно никакого отношения к **print** не имеет, а применяется к строкам. Однако часто форматирование используется в сочетании с вызовом функции **print().**

Форматирование может выполняться в так называемом старом стиле или с помощью строкового метода **format**. Старый стиль также называют Си-стилем, так как он схож с тем, как происходит вывод на экран в языке **С**. Здесь вместо трех комбинаций символов **%s, %d, %f** подставляются значения переменных **pupil, old, grade**. Буквы **s, d, f** обозначают типы данных – строку, целое число, вещественное число. Если бы требовалось подставить три строки, то во всех случаях использовалось бы сочетание **%s**.

>>> pupil = "Ben"

>>> old = 16

>>> grade = 9.2

>>> print("It's %s, %d. Level: %f" %

... (pupil, old, grade))

It's Ben, 16. Level: 9.200000

>>> print("This is a {0}. It's {1}."

... .format("ball", "red"))

This is a ball. It's red.

* 1. **Логические выражения**

Говоря на естественном языке (например, русском) мы обозначаем сравнения словами "равно", "больше", "меньше". В языках программирования используются специальные знаки, подобные тем, которые используются в математике: **>** (больше), **<** (меньше), **>=** (больше или равно), **<=** (меньше или равно), **==** (равно), **!=** (не равно).

Не путайте операцию присваивания значения переменной, обозначаемую в языке Python одиночным знаком "равно", и операцию сравнения (два знака "равно"). Присваивание и сравнение – разные операции.

Введем четвертый – **логический тип данных** (тип **bool**). Его также называют булевым. У этого типа всего два возможных значения: **True** (правда) и **False** (ложь).

>>> a = 10

>>> b = 5

>>> a + b > 14

True

>>> a != b

True

>>> a == b

False

>>> c = a == b

>>> a, b, c

(10, 5, False)

* 1. **Ветвление. Условный оператор**

Ход выполнения программы может быть линейным, то есть таким, когда выражения выполняются друг за другом, начиная с первого и заканчивая последним. Ни одна строка кода программы не пропускается.

Однако чаще в программах бывает не так. При выполнении кода, в зависимости от тех или иных условий, некоторые его участки могут быть опущены, в то время как другие – выполнены. Иными словами, в программе может присутствовать ветвление, которое реализуется условным оператором – особой конструкцией языка программирования.

if логическое\_выражение {

выражение 1;

выражение 2;

…

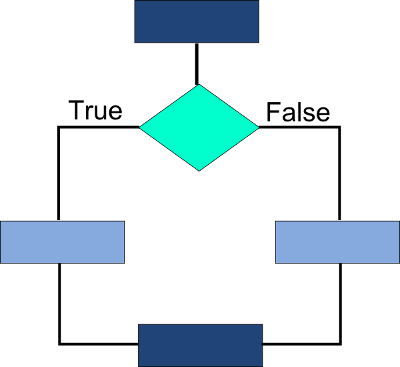
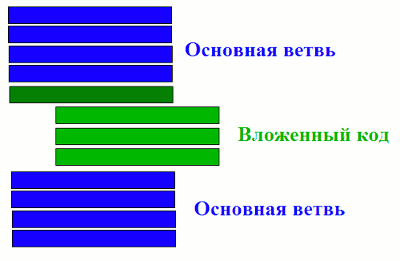
}

Конструкция **if** **логическое\_выражение** называется заголовком условного оператора. Выражения внутри фигурных скобок – телом условного оператора. Тело может содержать как множество выражений, так и всего одно или даже быть пустым.

**if** n < 100:

b = n + a

Структуру программы можно изобразить следующим образом:



* 1. **Циклы в программировании**

Циклы являются такой же важной частью структурного программирования, как условные операторы. С помощью циклов можно организовать повторение выполнения участков кода. Потребность в этом возникает довольно часто. Например, пользователь последовательно вводит числа, и каждое из них требуется добавлять к общей сумме. Или нужно вывести на экран квадраты ряда натуральных чисел и тому подобные задачи.

### **Цикл while**

"**While**" переводится с английского как "пока". Но не в смысле "до свидания", а в смысле "пока имеем это, делаем то". Можно сказать, **while** является универсальным циклом. Он присутствует во всех языках, поддерживающих структурное программирование, в том числе в Python. Его синтаксис обобщенно для всех языков можно выразить так:

while логическое\_выражение {

выражение 1;

…

выражение n;

}

В языке Python тоже есть цикл **for**. Но это не цикл со счетчиком. В Питоне он предназначен для перебора элементов последовательностей и других сложных объектов. Данный цикл и последовательности будут изучены в последующих уроках.

Для **while** наличие счетчика не обязательно. Представим, что надо вводить числа, пока переменная **total** больше нуля. Тогда код будет выглядеть так:

total = 100

**while** total > 0:

n = int(input())

total = total - n

**print**("Ресурс исчерпан")

### **Цикл for**

Цикл for в языке программирования Python предназначен для перебора элементов структур данных и некоторых других объектов. Это не цикл со счетчиком, каковым является **for** во многих других языках.

Что значит перебор элементов? Например, у нас есть список, состоящий из ряда элементов. Сначала берем из него первый элемент, затем второй, потом третий и так далее. С каждым элементом мы выполняем одни и те же действия в теле **for**. Нам не надо извлекать элементы по их индексам и заботится, на каком из них список заканчивается, и следующая итерация бессмысленна. Цикл **for** сам переберет и определит конец.

>>> spisok = [10, 40, 20, 30]

>>> for element in spisok:

... print(element + 2)

...

12

42

22

32

* 1. **Функции в программировании**

Функция в программировании представляет собой обособленный участок кода, который можно вызывать, обратившись к нему по имени, которым он был назван. При вызове происходит выполнение команд тела функции.

В языке программирования Python функции определяются с помощью оператора **def**. Рассмотрим код:

**def** countFood():

a = 5

b = 7

**print**("Всего", a+b, "шт.")

Это пример определения функции. Как и другие сложные инструкции вроде условного оператора и циклов функция состоит из заголовка и тела. Заголовок оканчивается двоеточием и переходом на новую строку. Тело имеет отступ.

**Вызов функции**

countFood()

* 1. **Локальные и глобальные переменные**

В программировании особое внимание уделяется концепции о локальных и глобальных переменных, а также связанное с ними представление об областях видимости. Соответственно, локальные переменные видны только в локальной области видимости, которой может выступать отдельно взятая функция. Глобальные переменные видны во всей программе. "Видны" – значит, известны, доступны. К ним можно обратиться по имени и получить связанное с ними значение.

result = 0

**def** rectangle():

a = float(input("Ширина: "))

b = float(input("Высота: "))

**global** result

result = a\*b

**def** triangle():

a = float(input("Основание: "))

h = float(input("Высота: "))

**global** result

result = 0.5 \* a \* h

figure = input("1-прямоугольник, 2-треугольник: ")

**if** figure == '1':

rectangle()

**elif** figure == '2':

triangle()

**print**("Площадь: %.2f" % result)

* 1. **Возврат значений из функции. Оператор return**

Функции могут передавать какие-либо данные из своих тел в основную ветку программы. Говорят, что функция возвращает значение. В большинстве языков программирования, в том числе Python, выход из функции и передача данных в то место, откуда она была вызвана, выполняется оператором **return**.

Если интерпретатор Питона, выполняя тело функции, встречает **return**, то он "забирает" значение, указанное после этой команды, и "уходит" из функции.

**def** cylinder():

r = float(input())

h = float(input())

*# площадь боковой поверхности цилиндра:*

side = 2 \* 3.14 \* r \* h

*# площадь одного основания цилиндра:*

circle = 3.14 \* r\*\*2

*# полная площадь цилиндра:*

full = side + 2 \* circle

**return** full

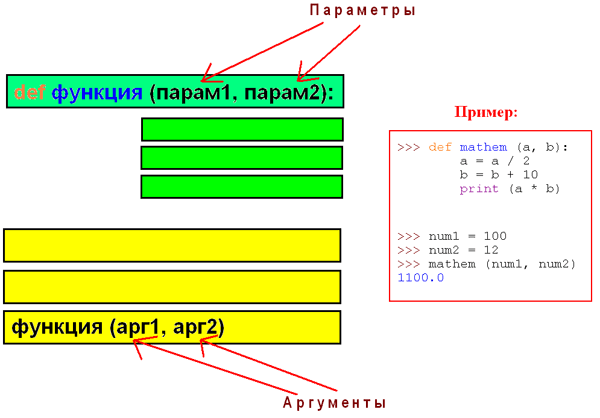
square = cylinder()

**print**(square)

* 1. **Параметры и аргументы функции**

В программировании функции могут не только возвращать данные, но также принимать их, что реализуется с помощью так называемых параметров, которые указываются в скобках в заголовке функции. Количество параметров может быть любым.

Параметры представляют собой локальные переменные, которым присваиваются значения в момент вызова функции. Конкретные значения, которые передаются в функцию при ее вызове, будем называть аргументами. Следует иметь в виду, что встречается иная терминология. Например, формальные параметры и фактические параметры. В Python же обычно все называют аргументами.



* 1. **Модули**

Встроенные в язык программирования функции доступны сразу. Чтобы их вызвать, не надо выполнять никаких дополнительных действий. Однако за время существования любого популярного языка на нем было написано столько функций и классов, которые оказались востребованными множеством программистов и в разных областях, что включить весь этот объем кода в сам язык если возможно, то нецелесообразно.

Чтобы разрешить проблему доступа к дополнительным возможностям языка, в программировании стало общепринятой практикой использовать так называемые модули, пакеты и библиотеки. Каждый модуль содержит коллекцию функций и классов, предназначенных для решения задач из определенной области. Так в модуле **math** языка Python содержатся математические функции, модуль **random** позволяет генерировать псевдослучайные числа, в модуле **datetime** содержатся классы для работы с датами и временем, модуль **sys** предоставляет доступ к системным переменным и т. д.

Количество модулей для языка Python огромно, что связано с популярностью языка. Часть модулей собрана в так называемую стандартную библиотеку. Стандартная она потому, что поставляется вместе с установочным пакетом. Однако существуют сторонние библиотеки. Они скачиваются и устанавливаются отдельно.

Для доступа к функционалу модуля, его надо импортировать в программу. После импорта интерпретатор "знает" о существовании дополнительных классов и функций и позволяет ими пользоваться. В Питоне импорт осуществляется командой **import**.

>>> import math

>>> math

<module 'math' (built-in)>

>>> dir(math)

['\_\_doc\_\_', '\_\_loader\_\_', '\_\_name\_\_',

'\_\_package\_\_', '\_\_spec\_\_', 'acos', 'acosh',

'asin', 'asinh', 'atan', 'atan2', 'atanh',

'ceil', 'copysign', 'cos', 'cosh',

'degrees', 'e', 'erf', 'erfc', 'exp',

'expm1', 'fabs', 'factorial', 'floor',

'fmod', 'frexp', 'fsum', 'gamma', 'gcd',

'hypot', 'inf', 'isclose', 'isfinite',

'isinf', 'isnan', 'ldexp', 'lgamma', 'log',

'log10', 'log1p', 'log2', 'modf',

'nan', 'pi', 'pow', 'radians', 'sin', 'sinh',

'sqrt', 'tan', 'tanh', 'trunc']

* 1. **Списки**

Список в Python – это встроенный тип (класс) данных, представляющий собой одну из разновидностей структур данных. Структуру данных можно представить как сложную единицу, объединяющую в себе группу более простых. Каждая разновидность структур данных имеет свои особенности. Список – это изменяемая последовательность произвольных элементов.

В большинстве других языков программирования есть такой широко используемый тип данных как массив. В Питоне такого встроенного типа нет. Однако списки – это по своей сути массивы за одним исключением. Составляющие массив элементы должны принадлежать одному типу данных, для python-списков такого ограничения нет.

>>> a = [12, 3.85, "black", -4]

>>> a

[12, 3.85, 'black', -4]

>>> a[3]

-4

>>> a[-3], a[-4]

(3.85, 12)

>>> a[0:2]

[12, 3.85]

>>> a[:3]

[12, 3.85, 'black']

>>> a[2:]

['black', -4]

>>> a[:]

[12, 3.85, 'black', -4]

>>> a.append('wood')

>>> a

[12, 4, 'black', -4, 'wood']

>>> a.insert(1, 'circle')

>>> a

[12, 'circle', 4, 'black', -4, 'wood']

>>> a.remove(4)

>>> a

[12, 'circle', 'black', -4, 'wood']

>>> a.pop()

'wood'

>>> a

[12, 'circle', 'black', -4]

* 1. **Функции первого класса и функция высшего порядка**

В информатике язык программирования имеет функции первого класса, если он рассматривает функции как объекты первого класса. В частности, это означает, что язык поддерживает передачу функций в качестве аргументов другим функциям, возврат их как результат других функций, присваивание их переменным или сохранение в структурах данных. В языках с функциями первого класса имена функций не имеют никакого специального статуса, они рассматриваются как обычные значения, тип которых является функциональным.

Функции первого класса являются неотъемлемой частью функционального программирования, в котором использование функций высшего порядка является стандартной практикой. Простым примером функции высшего порядка будет функция Map, которая принимает в качестве своих аргументов функцию и список и возвращает список после применения функции к каждому элементу списка. Чтобы язык программирования поддерживал Map, он должен поддерживать передачу функций как аргумента.

def f(x):

    return 3 \* x + 1

l = [1, 2, 3, 4, 5]

rez = map(f, l)

print(rez)

[4, 7, 10, 13, 16]

1. **Базовое программирование NAO на Python**

Среда разработки была выбрана Visual Studio Code. Для запуска проектов нужно нажать **F5** затем **Enter**.

Вся дополнительную информацию по работе фреймворка **naoqi** версии 2.8 и работе с **naoAPI** можно найти на [сайте разработчика](http://doc.aldebaran.com/2-8/naoqi/index.html).

Перед выполнением необходимо ознакомиться с таблицами [контактных и тактильных сенсоров](http://doc.aldebaran.com/2-8/family/nao_technical/contact-sensors_naov6.html) и [светодиодов](http://doc.aldebaran.com/2-8/naoqi/sensors/alleds.html). Для удобства все таблицы приведены в конце документа.

* 1. **Задание 1**

**Задание**: Включение и выключение глаз. При нажатии на передний датчик головы, левый или правый глаз (на выбор) загорается цветом(на выбор). При нажатии заднего датчика головы, глаз возвращается в стандартное(белое) состояние.

**Решение**: Создать бесконечный цикл, проверяющий статус сенсоров головы, и в зависимости от нажатого датчика окрашивать глаз в цвет, либо окрашивать в стандартный белый цвет.

**Схема программы:**

Основная схема

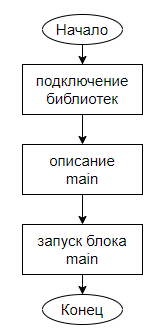
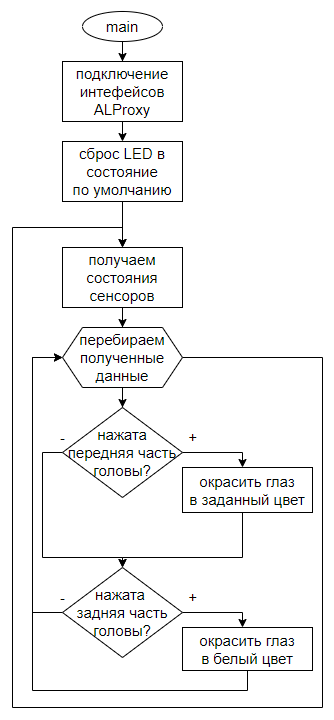


Схема блока main



**Код программы**:

# -\*- encoding: UTF-8 -\*-

import time                     #   подключение библиотеки time

from naoqi import ALProxy       #   Подключение ALProxy

from naoqi import ALBroker      #   Подключение ALBroker

from naoqi import ALModule      #   Подключение ALModule

def main(robot\_IP, robot\_PORT=9559):    #   Обьявление функции main. Функция принимает

два значения

                                        #   robot\_ip и robot\_port

    leds = ALProxy("ALLeds",robot\_IP,robot\_PORT)        #   Подключение интерфейса ALLeds

    touch = ALProxy("ALTouch",robot\_IP,robot\_PORT)      #   Подключение интерфейса ALTouch

    leds.reset("AllLeds")   #   Сброс LED по умолчанию

                            #   .reset("группа или конкретный светодиод")

    speed = 0.2             #   Скорость изменения цвета

    color = 0x00ff00ff      #   Добавление цвета в формате 0x00RRGGBB

    white = 0x00ffffff      #   где RR GG BB это 16-ричный код их цвета

    while True:             #   Бесконечный цикл

        status = touch.getStatus()  #   Получение состояний тактильных датчиков

        for e in status:            #   Цикл, проходящий по массиву status

                                    #   данные передаются в переменную e и с каждой итерацией

                                    #   берется следующий по массиву элемент

            if e[0] == "Head/Touch/Front" and e[1] == True:

#   Проверка значения состояния датчика "Head/Touch/Front".

#   Если true (датчик активирован), выполнить действие

                leds.fadeRGB("RightFaceLeds", color, speed)

#   Включение всех ледов правого глаза.

#   .fadeRGB("группа или конкретный светодиод", цвет, скорость изменения)

            if e[0] == "Head/Touch/Rear" and e[1] == True:

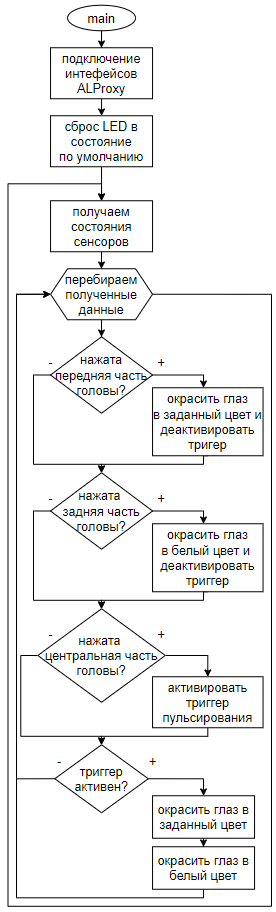
                leds.fadeRGB("RightFaceLeds", white, speed)

main("192.168.253.66", 9559)    #Вызов функции main. main("ip робота", порт 9559)

переменные используемые программой:

status – массив данных сенсоров, имеет вид:

[['Head', False, []], ['LArm', False, []], ['LLeg', False, []], ['RLeg', False, []], ['RArm', False, []], ['LHand', False, []], ['RHand', False, []], ['Head/Touch/Front', False, []], ['Head/Touch/Middle', False, []], ['Head/Touch/Rear', False, []], ['LFoot/Bumper/Left', False, []], ['LFoot/Bumper/Right', False, []], ['LHand/Touch/Back', False, []], ['LHand/Touch/Left', False, []], ['LHand/Touch/Right', False, []], ['RFoot/Bumper/Left', False, []], ['RFoot/Bumper/Right', False, []], ['RHand/Touch/Back', False, []], ['RHand/Touch/Left', False, []], ['RHand/Touch/Right', False, []]



e – элемент массива статус, имеет вид:

['Head', False, []]

e[0] = 'Head'

e[1] = False

* + 1. **Задание 1.1**

**Задание**: Добавить возможность моргания глаз при нажатии   
центрального датчика головы.

**Решение**: Добавить в бесконечный цикл, проверку нажатия  
центрального датчика головы и добавить блок   
обеспечивающий мерцание.

**Схема программы**:

Основная схема

**Код программы**:

# -\*- encoding: UTF-8 -\*-

import time

from naoqi import ALProxy

from naoqi import ALBroker

from naoqi import ALModule

def main(robot\_IP, robot\_PORT=9559):

    leds = ALProxy("ALLeds",robot\_IP,robot\_PORT)

    touch = ALProxy("ALTouch",robot\_IP,robot\_PORT)

    leds.reset("AllLeds")

    speed = 0.2

    color = 0x00ff00ff

    white = 0x00ffffff

    pulse = 0           #   Состояние пульсации глаз. 0-выклюбчена. 1-включена

    pulse\_speed = 0.5   #   Скорость пульсации глаз

    while True:

        status = touch.getStatus()

        for e in status:

            if e[0] == "Head/Touch/Front" and e[1] == True:

                leds.fadeRGB("RightFaceLeds", color, speed)

                pulse = 0

#   Выключение пульсации при нажатии "Head/Touch/Front"

            if e[0] == "Head/Touch/Rear" and e[1] == True:

                leds.fadeRGB("RightFaceLeds", white, speed)

                pulse = 0

#   Выключение пульсации при нажатии "Head/Touch/Rear"

            if e[0] == "Head/Touch/Middle" and e[1] == True:

                pulse = 1

#   Включение пульсации при нажатии "Head/Touch/Middle"

        if pulse == 1:

#   Проверка состояния значения переменной pulse

            leds.fadeRGB("RightFaceLeds", color, pulse\_speed)

#   Включение светодиода цветом color

            leds.fadeRGB("RightFaceLeds", white, pulse\_speed)

#   Включение светодиода цветом white

main("192.168.253.22", 9559)    #Вызов функции main. main("ip робота", порт 9559)

* + 1. **Задание 1.2**

**Задание**: Изменить программу, чтобы во всех операциях использовались оба глаза вместо одного.

**Решение**: Заменить группу светодиодов на “FaceLeds”.

**Код программы**:

# -\*- encoding: UTF-8 -\*-

import time

from naoqi import ALProxy

from naoqi import ALBroker

from naoqi import ALModule

def main(robot\_IP, robot\_PORT=9559):

    leds = ALProxy("ALLeds",robot\_IP,robot\_PORT)

    touch = ALProxy("ALTouch",robot\_IP,robot\_PORT)

    leds.reset("AllLeds")

    speed = 0.2

    color = 0x00ff00ff

    pulse = 0

    pulse\_speed = 0.5

    while True:

        status = touch.getStatus()

        for e in status:

            if e[0] == "Head/Touch/Front" and e[1] == True:

                leds.fadeRGB("FaceLeds", color, speed)

#   для подключения второго глаза была изменена группа с

#   "RightFaceLeds" на "FaceLeds", что позоляет изменять цвет всех глаз

                pulse = 0

            if e[0] == "Head/Touch/Rear" and e[1] == True:

                leds.fadeRGB("FaceLeds", 0x00ffffff, speed)

                pulse = 0

            if e[0] == "Head/Touch/Middle" and e[1] == True:

                pulse = 1

        if pulse == 1:

            leds.fadeRGB("FaceLeds", color, pulse\_speed)

            leds.fadeRGB("FaceLeds", 0x00ffffff, pulse\_speed)

main("192.168.253.22", 9559)    #Вызов функции main. main("ip робота", порт 9559)

* 1. **Задание 2**

**Задание**: Светофор. Используя три датчика касания расположенных на голове, активировать все светодиоды 3 разными цветами(красный, желтый, зеленый)

**Решение**: Использовать группу AllLeds.

**Схема программы**:

Схожа с заданием 1, необходимо использовать новую группу для использования ВСЕХ светодиодов.

**Код программы**:

# -\*- encoding: UTF-8 -\*-

import time

from naoqi import ALProxy

from naoqi import ALBroker

from naoqi import ALModule

def main(robot\_IP, robot\_PORT=9559):

    leds = ALProxy("ALLeds",robot\_IP,robot\_PORT)

    touch = ALProxy("ALTouch",robot\_IP,robot\_PORT)

    leds.reset("AllLeds")

    green = 0x0000ff00

    yellow = 0x00ffff00

    red = 0x00ff0000

    while True:

        status = touch.getStatus()

        for e in status:

            if e[0] == "Head/Touch/Front" and e[1] == True:

#   Проверяет состояние сенсора Head/Touch/Front (передняя часть головы)

                leds.fadeRGB("AllLeds", green,0.2)

#   Окрашивает все цветные светодиоды в ЗЕЛЕНЫЙ цвет (0x0000ff00)

            if e[0] == "Head/Touch/Middle" and e[1] == True:

#   Проверяет состояние сенсора Head/Touch/Middle (центральная часть головы)

                leds.fadeRGB("AllLeds", yellow,0.2)

#   Окрашивает все цветные светодиоды в ЖЕЛТЫЙ цвет (0x00ffff00)

            if e[0] == "Head/Touch/Rear" and e[1] == True:

#   Проверяет состояние сенсора Head/Touch/Rear (задняя часть головы)

                leds.fadeRGB("AllLeds", red,0.2)

#   Окрашивает все цветные светодиоды в КРАСНЫЙ цвет (0x00ff0000)

main("192.168.253.22", 9559)

* + 1. **Задание 2.1**

**Задание**: Добавить настроения. При нажатии одного из датчиков головы, должны загораться по 2 сегмента(светодиода) на каждом глазу, создавая таким образом 3 разных комбинации настроения  
( \ / ) ( / \ ) ( - - ).

**Решение**: Окрашивать конкретные сегменты глаза. FaceLed[x]. x-номер необходимого сектора.

**Схема программы**:

Схожа с заданием 1, только вместо всех светодиодов, но дополнительно окрашиваются необходимые сегменты в ручную.

**Код программы**:

# -\*- encoding: UTF-8 -\*-

import time

from naoqi import ALProxy

from naoqi import ALBroker

from naoqi import ALModule

def main(robot\_IP, robot\_PORT=9559):

    leds = ALProxy("ALLeds",robot\_IP,robot\_PORT)

#   Подключение интерфейса для работы с диодами

    touch = ALProxy("ALTouch",robot\_IP,robot\_PORT)

#   Подключение интерфейса для работы с тактильными сенсорами

    leds.reset("AllLeds")

    green = 0x0000ff00          #   Добавление цвета в формате 0x00RRGGBB

    yellow = 0x00ffff00         #   где RR GG BB это 16-ричный код их цвета

    red = 0x00ff0000            #

    mood\_bg = 0x00000000

    speed = 0.1

    while True:

        status = touch.getStatus()

#   Получает массив с состояниями тактильных сенсоров

        for e in status:

#   Цикл, переберающий массив состояний тактильных сенсоров

            if e[0] == "Head/Touch/Front" and e[1] == True:

#   Проверяет состояние сенсора Head/Touch/Front (передняя часть головы)

                leds.fadeRGB("AllLeds", green, speed)

#   Окрашивает все цветные светодиоды в ЗЕЛЕНЫЙ цвет (0x0000ff00)

                leds.fadeRGB("FaceLed0", mood\_bg, speed)

                leds.fadeRGB("FaceLed1", green, speed)

                leds.fadeRGB("FaceLed2", mood\_bg, speed)

                leds.fadeRGB("FaceLed3", mood\_bg, speed)

                leds.fadeRGB("FaceLed4", mood\_bg, speed)

                leds.fadeRGB("FaceLed5", green, speed)

                leds.fadeRGB("FaceLed6", mood\_bg, speed)

                leds.fadeRGB("FaceLed7", mood\_bg, speed)

            if e[0] == "Head/Touch/Middle" and e[1] == True:

#   Проверяет состояние сенсора Head/Touch/Middle (центральная часть головы)

                leds.fadeRGB("AllLeds", yellow, speed)

#   Окрашивает все цветные светодиоды в ЖЕЛТЫЙ цвет (0x00ffff00)

                leds.fadeRGB("FaceLed0", mood\_bg, speed)

                leds.fadeRGB("FaceLed1", mood\_bg, speed)

                leds.fadeRGB("FaceLed2", yellow, speed)

                leds.fadeRGB("FaceLed3", mood\_bg, speed)

                leds.fadeRGB("FaceLed4", mood\_bg, speed)

                leds.fadeRGB("FaceLed5", mood\_bg, speed)

                leds.fadeRGB("FaceLed6", yellow, speed)

                leds.fadeRGB("FaceLed7", mood\_bg, speed)

            if e[0] == "Head/Touch/Rear" and e[1] == True:

#   Проверяет состояние сенсора Head/Touch/Rear (задняя часть головы)

                leds.fadeRGB("AllLeds", red, speed)

#   Окрашивает все цветные светодиоды в КРАСНЫЙ цвет (0x00ff0000)

                leds.fadeRGB("FaceLed0", mood\_bg, speed)

                leds.fadeRGB("FaceLed1", mood\_bg, speed)

                leds.fadeRGB("FaceLed2", mood\_bg, speed)

                leds.fadeRGB("FaceLed3", red, speed)

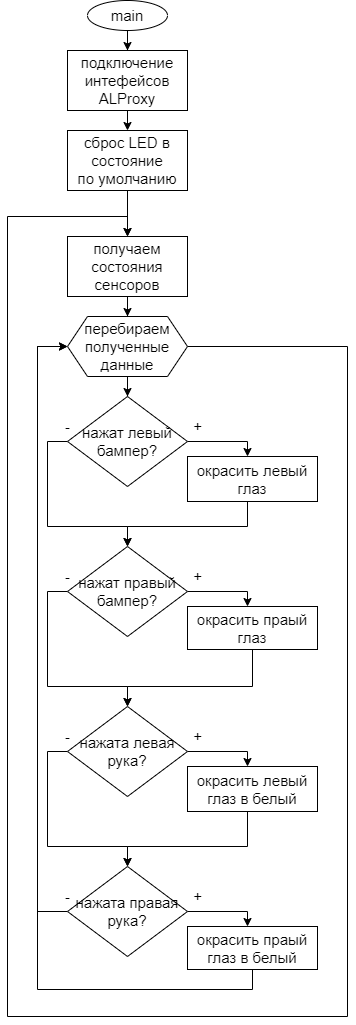
                leds.fadeRGB("FaceLed4", mood\_bg, speed)

                leds.fadeRGB("FaceLed5", mood\_bg, speed)

                leds.fadeRGB("FaceLed6", mood\_bg, speed)

                leds.fadeRGB("FaceLed7", red, speed)

main("192.168.253.22", 9559)

* 1. **Задание 3**

**Задание**: При нажатии на бампер, глаз загорается нужным  
цветом на соответствующей стороне(правый-правый,  
левый-левый). При нажатии руки, глаз меняет цвет на белый.

**Решение**: Сделать 4 проверки сенсоров, по 2 проверки на  
ноги и на руки. При активации одного из них, либо   
окрашивать, либо возвращать начальный цвет глаза.

**Схема программы**:

**Код программы**:

# -\*- encoding: UTF-8 -\*-

import time

from naoqi import ALProxy

from naoqi import ALBroker

from naoqi import ALModule

def main(robot\_IP, robot\_PORT=9559):

    leds = ALProxy("ALLeds",robot\_IP,robot\_PORT)

    touch = ALProxy("ALTouch",robot\_IP,robot\_PORT)

    leds.reset("AllLeds")

    color1 = 0x00ffff00

    color2 = 0x00ff00ff

    white = 0x00ffffff

    speed = 0.2

    while True:

        status = touch.getStatus()

        for e in status:

            if e[0] == "LFoot/Bumper/Left" and e[1] == True:

                leds.fadeRGB("LeftFaceLeds", color1, speed)

            elif e[0] == "LFoot/Bumper/Right" and e[1] == True:

                leds.fadeRGB("LeftFaceLeds", color1, speed)

            if e[0] == "RFoot/Bumper/Left" and e[1] == True:

                leds.fadeRGB("RightFaceLeds", color2, speed)

            elif e[0] == "RFoot/Bumper/Right" and e[1] == True:

                leds.fadeRGB("RightFaceLeds", color2, speed)

            if e[0] == "LArm" and e[1] == True:

                leds.fadeRGB("LeftFaceLeds", white, speed)

            if e[0] == "RArm" and e[1] == True:

                leds.fadeRGB("RightFaceLeds", white, speed)

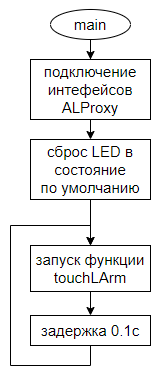
main("192.168.253.22", 9559)

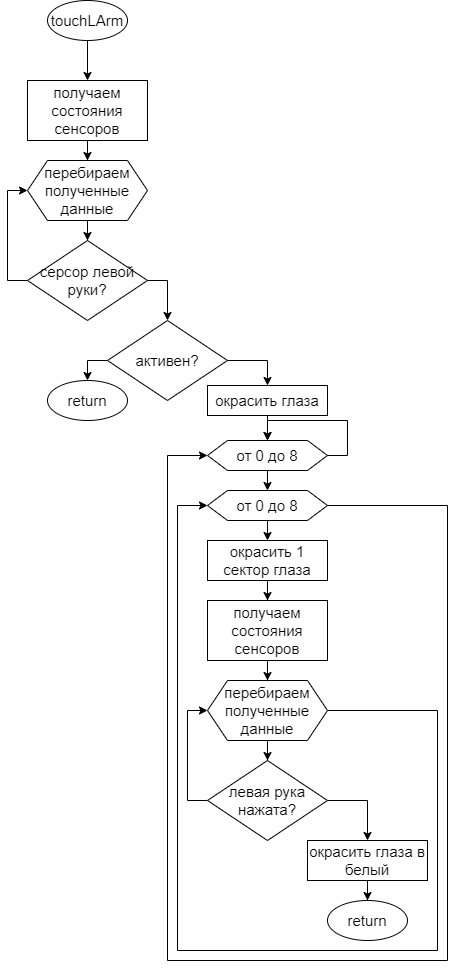
* 1. **Задание 4**

**Задание**: Бегущая радуга. При нажатии левой руки. Поочередно по часовой стрелке загорается каждый из восьми сегментов глаза разным цветом, создавая эффект бегущего огня. Повторное нажатие руки выключает радугу и переводит глаза в начальный цвет.

**Решение**: Необходимо создать новую функцию, проворящую состояние датчика руки и  
включающую \ выключающую бегущие огни.

**Схема программы**:





**Код программы**:

# -\*- encoding: UTF-8 -\*-

import time

from naoqi import ALProxy

from naoqi import ALBroker

from naoqi import ALModule

rainbow=[       #   Обьявление массива кодов разных цветов

    0x00ff0000, 0x00ff7f00, 0x00ffff00, 0x007fff00, 0x0000ffff, 0x000000ff, 0x007f00ff, 0x00ff00ff,

    0x00ff0000, 0x00ff7f00, 0x00ffff00, 0x007fff00, 0x0000ffff, 0x000000ff, 0x007f00ff, 0x00ff00ff]

white = 0x00ffffff

speed = 0.2

rotate\_speed = 0.01

def main(robot\_IP, robot\_PORT=9559):

    leds = ALProxy("ALLeds",robot\_IP,robot\_PORT)

    touch = ALProxy("ALTouch",robot\_IP,robot\_PORT)

    leds.reset("AllLeds")

    while True:

        touchLArm(leds,touch,"LArm","LeftFaceLed{}")

        time.sleep(0.1)                 #   Ожидаем 0.1 секунду

def touchLArm(leds,touch,sensor,led):

#   Обьявление функции touchLArm, с получаемыми параметрами leds,touch,sensor,led

             # sensor = "LArm"

             # led = "LeftFaceLed{}"

    while 1:

        status = touch.getStatus()

        for e in status:

            if e[0] == sensor:

                if e[1]==True:

                    for i in range(8):      #   окраска ледов в цикле, постепенно

                            leds.fadeRGB(led.format(8-i), rainbow[i], speed)

                else:

                    return

#   выход и завершение функции в случае отсутствия активных датчиков

                time.sleep(1)

                while 1:

                    for i in range(8):

#   двойной цикл для создания бегущего огня

                        for j in range(8):

                            leds.fadeRGB(led.format(j+1), rainbow[i+j], rotate\_speed)

                            status = touch.getStatus()

                            for e in status:

#   в случае повторного нажатия на датчик, запускает

цикл окраски глаза в обычный цвет и выхода из функции

                                if e[0] == sensor and e[1]==True:

                                    for i in range(8):

                                        leds.fadeRGB(led.format(8-i), white,speed)

                                    return

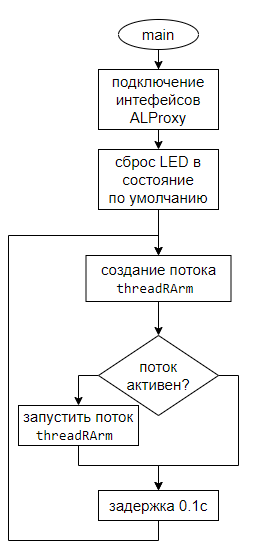
main("192.168.253.66", 9559)

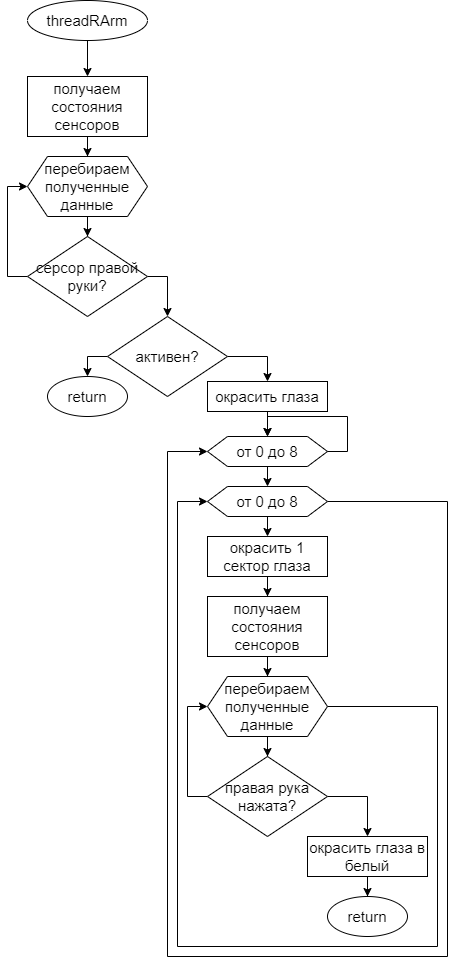
* + 1. **Задание 4.1**

**Задание**: Создать аналогичную программу, только для правой руки и глаза.

**Решение**: Необходимо поменять группы сенсоров и светодиодов. Однако задание можно усложнить, добавив многопоточность, которая пригодится в задании 4.2.

**Схема программы**:





**Код программы**:

# -\*- encoding: UTF-8 -\*-

import time

import threading                    #   Подключение библиотеки threadnig

from naoqi import ALProxy

from naoqi import ALBroker

from naoqi import ALModule

from threading import Thread        #   Подключение модуля Thread из библиотеки threadnig

rainbow=[0x00ff0000, 0x00ff7f00, 0x00ffff00, 0x007fff00, 0x0000ffff, 0x000000ff, 0x007f00ff, 0x00ff00ff,

         0x00ff0000, 0x00ff7f00, 0x00ffff00, 0x007fff00, 0x0000ffff, 0x000000ff, 0x007f00ff, 0x00ff00ff]

white = 0x00ffffff

speed = 0.2

rotate\_speed = 0.01

def main(robot\_IP, robot\_PORT=9559):

    leds = ALProxy("ALLeds",robot\_IP,robot\_PORT)

    touch = ALProxy("ALTouch",robot\_IP,robot\_PORT)

    leds.reset("AllLeds")

    global rArmIsActive         #   переменная - статус потока

    rArmIsActive = True         #   True - поток не запущен, False-запущен

    while True:

        threadRArm = Thread(target=touchRArm, args=(leds,touch,"RArm","RightFaceLed{}"))

#   создание потока для правой руки

        #   target = необходимая функция

        #   args = аргументы, передаваемые в функцию

        if rArmIsActive:

            threadRArm.start()      #   если поток не запущен, запустить его

        time.sleep(0.1)

def touchRArm(leds,touch,sensor,led):

    global rArmIsActive

    rArmIsActive = False

    while 1:

        status = touch.getStatus()

        for e in status:

            if e[0] == sensor:

                if e[1]==True:

                    for i in range(8):

                            leds.fadeRGB(led.format(i+1), rainbow[i], speed)

                else:

                    rArmIsActive = True

                    return

                time.sleep(1)

                while 1:

                    for i in range(8):

                        for j in range(8):

                            leds.fadeRGB(led.format(8-j), rainbow[i+j], rotate\_speed)

                            status = touch.getStatus()

                            for e in status:

                                if e[0] == sensor and e[1]==True:

                                    for i in range(8):

                                        leds.fadeRGB(led.format(i+1), white,speed)

                                    rArmIsActive = False

                                    return

main("192.168.252.226", 9559)

* + 1. **Задание 4.2**

**Задание**: Объединить пункты 4.1 и 4.2

**Решение**: Для совместной работы нескольких функций необходимо воспользоваться многопоточностью.

**Схема программы**:

Все так же, только появляется второй поток.

**Код программы**:

# -\*- encoding: UTF-8 -\*-

import time

import threading

from naoqi import ALProxy

from naoqi import ALBroker

from naoqi import ALModule

from threading import Thread

rainbow=[0x00ff0000, 0x00ff7f00, 0x00ffff00, 0x007fff00, 0x0000ffff, 0x000000ff, 0x007f00ff, 0x00ff00ff,

         0x00ff0000, 0x00ff7f00, 0x00ffff00, 0x007fff00, 0x0000ffff, 0x000000ff, 0x007f00ff, 0x00ff00ff]

white = 0x00ffffff

speed = 0.2

rotate\_speed = 0.01

def main(robot\_IP, robot\_PORT=9559):

    leds = ALProxy("ALLeds",robot\_IP,robot\_PORT)

    touch = ALProxy("ALTouch",robot\_IP,robot\_PORT)

    leds.reset("AllLeds")

    global rArmIsActive, lArmIsActive

    rArmIsActive, lArmIsActive = True,True

    while True:

        threadRArm = Thread(target=touchRArm, args=(leds,touch,"RArm","RightFaceLed{}"))

        threadLArm = Thread(target=touchLArm, args=(leds,touch,"LArm","LeftFaceLed{}"))

        #   создание двух потоков

        if lArmIsActive:

            threadLArm.start()

                                #   В случае если не запущен поток, запускает его

        if rArmIsActive:

            threadRArm.start()

        time.sleep(0.1)

def touchRArm(leds,touch,sensor,led):

    global rArmIsActive

    rArmIsActive = False

    while 1:

        status = touch.getStatus()

        for e in status:

            if e[0] == sensor:

                if e[1]==True:

                    for i in range(8):

                            leds.fadeRGB(led.format(i+1), rainbow[i], speed)

                else:

                    rArmIsActive = True

                    return

                time.sleep(1)

                while 1:

                    for i in range(8):

                        for j in range(8):

                            leds.fadeRGB(led.format(8-j), rainbow[i+j], rotate\_speed)

                            status = touch.getStatus()

                            for e in status:

                                if e[0] == sensor and e[1]==True:

                                    for i in range(8):

                                        leds.fadeRGB(led.format(i+1), white,speed)

                                    rArmIsActive = True

                                    return

def touchLArm(leds,touch,sensor,led):

    global lArmIsActive

    lArmIsActive = False

    while 1:

        status = touch.getStatus()

        for e in status:

            if e[0] == sensor:

                if e[1]==True:

                    for i in range(8):

                            leds.fadeRGB(led.format(8-i), rainbow[i], speed)

                else:

                    lArmIsActive = True

                    return

                time.sleep(1)

                while 1:

                    for i in range(8):

                        for j in range(8):

                            leds.fadeRGB(led.format(j+1), rainbow[i+j], rotate\_speed)

                            status = touch.getStatus()

                            for e in status:

                                if e[0] == sensor and e[1]==True:

                                    for i in range(8):

                                        leds.fadeRGB(led.format(8-i), white,speed)

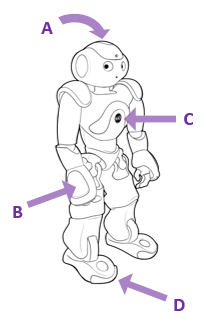
                                    lArmIsActive = True

                                    return

main("192.168.252.226", 9559)

1. **Дополнения**
   1. **Контактные и тактильные сенсоры**

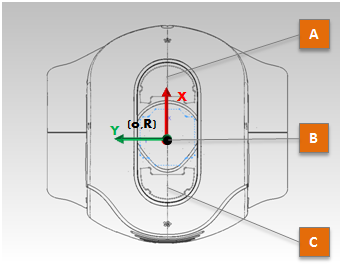
4 набора сенсоров. Всего 14 различных сенсоров



| **Часть** | **Название** | **Часть** | **Название** |
| --- | --- | --- | --- |
| **A** | Сенсоры головы | **B** | Сенсоры рук |
| **C** | Кнопка на груди | **D** | Сенсоры ног |

* + 1. **Сенсоры головы**

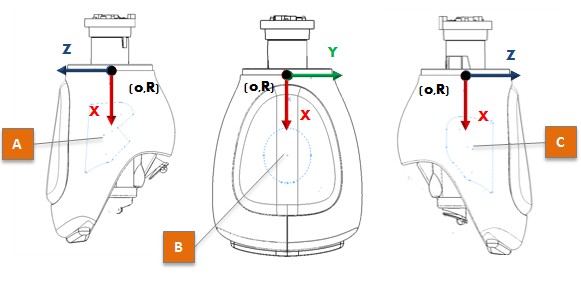
3 датчика на голове



| **Часть** | **Название** |
| --- | --- |
| A | Head/Touch/Front |
| B | Head/Touch/Middle |
| C | Head/Touch/Rear |

* + 1. **Сенсоры рук**

2 набора по 3 датчика на каждую руку



| **Часть** | **Название** | **Часть** | **Название** |
| --- | --- | --- | --- |
| **A** | LHand/Touch/Left | **A** | RHand/Touch/Left |
| **B** | LHand/Touch/Back | **B** | RHand/Touch/Back |
| **C** | LHand/Touch/Right | **C** | RHand/Touch/Right |

* + 1. **Кнопка на груди**

1 кнопка на груди

| **Название** |
| --- |
| ChestBoard/Button |

* + 1. **Сенсоры ног**

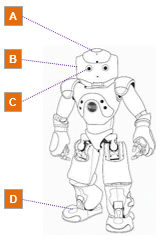
2 набора по 2 датчика на кончиках ступней

|  |  |
| --- | --- |
| ../../_images/hardware_bumper_1.png | ../../_images/hardware_bumper_2.png |
|

| **Название** | **Название** |
| --- | --- |
| LFoot/Bumper/Left | RFoot/Bumper/Left |
| LFoot/Bumper/Right | RFoot/Bumper/Right |

* 1. **Таблица светодиодов**

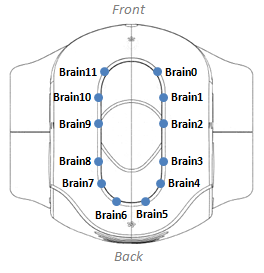
4 набора светодиодов



| **Часть** | **Название** | **Кол-во** | **Описание** |
| --- | --- | --- | --- |
| **A** | Тактильные сенсоры головы | 2x6 | 16 уровней яркости белого цвета |
| **B** | Глаза | 2x8 | полный RGB набор |
| **C** | Уши | 2x10 | 16 уровней яркости синего цвета |
| **D** | Ноги | 2 | полный RGB набор |

* + 1. **Тактильные сенсоры головы**

2 набора по 6 **белых** светодиодов



**Право**

| **Короткое название** | **Полное название** |
| --- | --- |
| Brain0 | Head/Led/Front/Right/1/Actuator/Value |
| Brain1 | Head/Led/Front/Right/0/Actuator/Value |
| Brain2 | Head/Led/Middle/Right/0/Actuator/Value |
| Brain3 | Head/Led/Rear/Right/0/Actuator/Value |
| Brain4 | Head/Led/Rear/Right/1/Actuator/Value |
| Brain5 | Head/Led/Rear/Right/2/Actuator/Value |

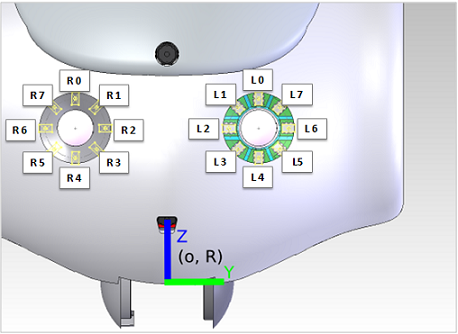
**Лево**

| **Короткое название** | **Полное название** |
| --- | --- |
| Brain6 | Head/Led/Rear/Left/2/Actuator/Value |
| Brain7 | Head/Led/Rear/Left/1/Actuator/Value |
| Brain8 | Head/Led/Rear/Left/0/Actuator/Value |
| Brain9 | Head/Led/Middle/Left/0/Actuator/Value |
| Brain10 | Head/Led/Front/Left/0/Actuator/Value |
| Brain11 | Head/Led/Front/Left/1/Actuator/Value |

| **Группы** |
| --- |
| BrainLeds |
| BrainLedsBack  BrainLedsMiddle  BrainLedsFront |
| BrainLedsLeft  BrainLedsRight |

* + 1. **Глаза**

2 набора по 8 **RGB** светодиодов



**Право**

| **Часть** | **Название группы** | **Короткое название** | **Actuator Value** |
| --- | --- | --- | --- |
| R 0 | FaceLedRight0 | RightFaceLed8 | Face/Led/[Color]/Right/315Deg |
| R 1 | FaceLedRight1 | RightFaceLed7 | Face/Led/[Color]/Right/270Deg |
| R 2 | FaceLedRight2 | RightFaceLed6 | Face/Led/[Color]/Right/225Deg |
| R 3 | FaceLedRight3 | RightFaceLed5 | Face/Led/[Color]/Right/180Deg |
| R 4 | FaceLedRight4 | RightFaceLed4 | Face/Led/[Color]/Right/135Deg |
| R 5 | FaceLedRight5 | RightFaceLed3 | Face/Led/[Color]/Right/90Deg |
| R 6 | FaceLedRight6 | RightFaceLed2 | Face/Led/[Color]/Right/45Deg |
| R 7 | FaceLedRight7 | RightFaceLed1 | Face/Led/[Color]/Right/0Deg |

**Лево**

| **Часть** | **Название группы** | **Короткое название** | **Actuator Value** |
| --- | --- | --- | --- |
| L 0 | FaceLedLeft0 | LeftFaceLed1 | Face/Led/[Color]/Left/0Deg |
| L 1 | FaceLedLeft1 | LeftFaceLed2 | Face/Led/[Color]/Left/45Deg |
| L 2 | FaceLedLeft2 | LeftFaceLed3 | Face/Led/[Color]/Left/90Deg |
| L 3 | FaceLedLeft3 | LeftFaceLed4 | Face/Led/[Color]/Left/135Deg |
| L 4 | FaceLedLeft4 | LeftFaceLed5 | Face/Led/[Color]/Left/180Deg |
| L 5 | FaceLedLeft5 | LeftFaceLed6 | Face/Led/[Color]/Left/225Deg |
| L 6 | FaceLedLeft6 | LeftFaceLed7 | Face/Led/[Color]/Left/270Deg |
| L 7 | FaceLedLeft7 | LeftFaceLed8 | Face/Led/[Color]/Left/315Deg |

**Группы**

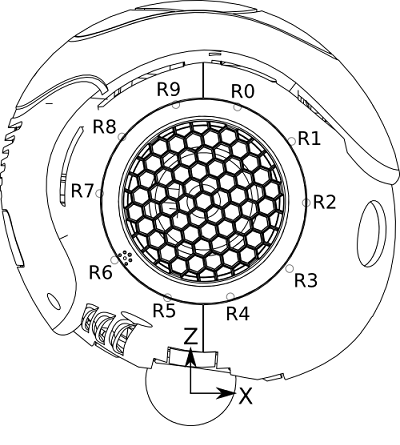
| **Группа** | **Группа** | *= короткое название* | **Группа** | *= короткое название* |
| --- | --- | --- | --- | --- |
| FaceLed0 | FaceLedRight0 | RightFaceLed8 | FaceLedLeft0 | LeftFaceLed1 |
| FaceLed1 | FaceLedRight1 | RightFaceLed7 | FaceLedLeft1 | LeftFaceLed2 |
| FaceLed2 | FaceLedRight2 | RightFaceLed6 | FaceLedLeft2 | LeftFaceLed3 |
| FaceLed3 | FaceLedRight3 | RightFaceLed5 | FaceLedLeft3 | LeftFaceLed4 |
| FaceLed4 | FaceLedRight4 | RightFaceLed4 | FaceLedLeft4 | LeftFaceLed5 |
| FaceLed5 | FaceLedRight5 | RightFaceLed3 | FaceLedLeft5 | LeftFaceLed6 |
| FaceLed6 | FaceLedRight6 | RightFaceLed2 | FaceLedLeft6 | LeftFaceLed7 |
| FaceLed7 | FaceLedRight7 | RightFaceLed1 | FaceLedLeft7 | LeftFaceLed8 |

| **Groups** | | |
| --- | --- | --- |
| FaceLeds | RightFaceLeds | LeftFaceLeds |
| FaceLedsBottom  FaceLedsExternal  FaceLedsInternal  FaceLedsTop | FaceLedsRightBottom  FaceLedsRightExternal  FaceLedsRightInternal  FaceLedsRightTop | FaceLedsLeftBottom  FaceLedsLeftExternal  FaceLedsLeftInternal  FaceLedsLeftTop |
|  | RightFaceLedsBlue  RightFaceLedsGreen  RightFaceLedsRed | LeftFaceLedsBlue  LeftFaceLedsGreen  LeftFaceLedsRed |

* + 1. **Уши**

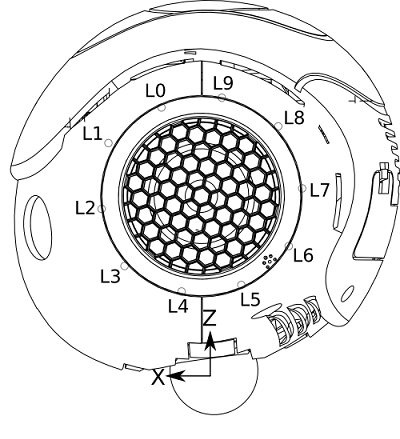
2 набора по 10 **синих** светодиодов

**Правое ухо**



| **Led Ref** | **Short Name** | **Actuator Value** |
| --- | --- | --- |
| R 0 | RightEarLed1 | Ears/Led/Right/0Deg/Actuator/Value |
| R 1 | RightEarLed2 | Ears/Led/Right/36Deg/Actuator/Value |
| R 2 | RightEarLed3 | Ears/Led/Right/72Deg/Actuator/Value |
| R 3 | RightEarLed4 | Ears/Led/Right/108Deg/Actuator/Value |
| R 4 | RightEarLed5 | Ears/Led/Right/144Deg/Actuator/Value |
| R 5 | RightEarLed6 | Ears/Led/Right/180Deg/Actuator/Value |
| R 6 | RightEarLed7 | Ears/Led/Right/216Deg/Actuator/Value |
| R 7 | RightEarLed8 | Ears/Led/Right/252Deg/Actuator/Value |
| R 8 | RightEarLed9 | Ears/Led/Right/288Deg/Actuator/Value |
| R 9 | RightEarLed10 | Ears/Led/Right/324Deg/Actuator/Value |

**Левое ухо**

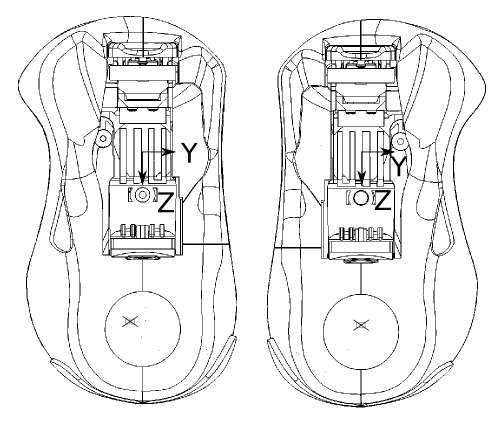


| **Led Ref** | **Short Name** | **Actuator Value** |
| --- | --- | --- |
| L 0 | LeftEarLed1 | Ears/Led/Left/0Deg/Actuator/Value |
| L 1 | LeftEarLed2 | Ears/Led/Left/36Deg/Actuator/Value |
| L 2 | LeftEarLed3 | Ears/Led/Left/72Deg/Actuator/Value |
| L 3 | LeftEarLed4 | Ears/Led/Left/108Deg/Actuator/Value |
| L 4 | LeftEarLed5 | Ears/Led/Left/144Deg/Actuator/Value |
| L 5 | LeftEarLed6 | Ears/Led/Left/180Deg/Actuator/Value |
| L 6 | LeftEarLed7 | Ears/Led/Left/216Deg/Actuator/Value |
| L 7 | LeftEarLed8 | Ears/Led/Left/252Deg/Actuator/Value |
| L 8 | LeftEarLed9 | Ears/Led/Left/288Deg/Actuator/Value |
| L 9 | LeftEarLed10 | Ears/Led/Left/324Deg/Actuator/Value |

| **Groups** | |
| --- | --- |
| EarLeds | |
| RightEarLeds | LeftEarLeds |
| RightEarLedsBack  RightEarLedsFront | LeftEarLedsBack  LeftEarLedsFront |
| RightEarLedsEven  RightEarLedsOdd | LeftEarLedsEven  LeftEarLedsOdd |

* + 1. **Ноги**

По 1 **RGB** светодиоду на каждой ноге



| **Led Ref** | **Group Name** | **Short Name** | **Actuator Value** |
| --- | --- | --- | --- |
| Foot Right | RightFootLeds | RightFootLedsBlue RightFootLedsGreen RightFootLedsRed | RFoot/Led/Blue/Actuator/Value  RFoot/Led/Green/Actuator/Value  RFoot/Led/Red/Actuator/Value |
| Foot Left | LeftFootLeds | LeftFootLedsBlue LeftFootLedsGreen LeftFootLedsRed | LFoot/Led/Blue/Actuator/Value  LFoot/Led/Green/Actuator/Value  LFoot/Led/Red/Actuator/Value |
| Both Foot | FeetLeds |  |  |

* + 1. **Кнопка на груди**

| **Groups** |
| --- |
| ChestLeds |

| **Short Name** | **Name** |
| --- | --- |
| ChestLedsBlue | ChestBoard/Led/Blue/Actuator/Value |
| ChestLedsGreen | ChestBoard/Led/Green/Actuator/Value |
| ChestLedsRed | ChestBoard/Led/Red/Actuator/Value |

* + 1. **Дополнительно**

*Все светодиоды*

| **Groups** |
| --- |
| AllLeds |
| AllLedsBlue  AllLedsGreen  AllLedsRed |