Занятие 27

НЕ КОНСПЕКТ! Материал для ознакомления, необходимо прочитать и выполнить в тетради тренировочные задания.

Как мыслить алгоритмами

Что такое алгоритм? Это набор инструкций или шагов, которые выполняются для решения определённой задачи.

Алгоритм — не просто список действий, а чёткое, разбитое на мелкие шаги описание: что и в какой последовательности нужно сделать.

Например, вам хочется горячего шоколада. Давайте посмотрим на это как на алгоритмическую задачу.

Список действий

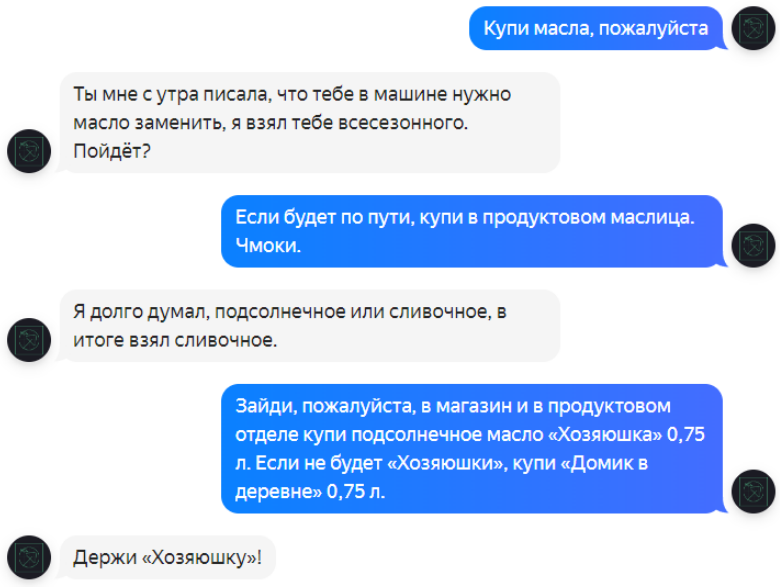
- Достать молоко
- Найти шоколадный порошок
- Открыть холодильник
- Поискать чашку

Алгоритм

1. Взять чашку
2. Открыть шкаф
3. Взять пакетик шоколадного порошка
4. Насыпать шоколадный порошок в чашку
5. Открыть холодильник
6. Взять молоко
7. Налить молоко в чашку
8. Перемешать
9. Подогреть
10. Горячий шоколад готов!

А что такое тогда «алгоритмическое мышление»? Это похоже на специальные очки, через которые вы смотрите на мир: вы видите, как большие задачи можно разбить на подзадачи, и сразу выстраиваете последовательный план действий.

Представим, что мама купила продукты, но вот незадача: забыла о подсолнечном масле. Давайте напишем алгоритм для сына, который как раз едет к ней в гости, — пусть прихватит его по дороге.



С третьей попытки мама получила именно то, что нужно. Алгоритм сформулирован отлично! Как видите, при составлении алгоритмов нужно придерживаться определённых принципов, иначе есть риск получить на выходе что-то не то. Вот эти принципы.

Принципы создания алгоритмов

1. **Точность**: в описании действий не должно быть двусмысленности. Например: «взять подсолнечное масло определённой марки, такого-то объёма».
2. **Последовательность**: шаги алгоритма должны быть расположены в правильном порядке, один за другим. Например: «сначала посмотреть в магазине, есть ли подсолнечное масло, а потом купить».
3. **Вариативность**: нужно предусмотреть неожиданности и описать, что делать в этих случаях. Например: «если не будет товара А, купи товар Б».
4. **Минимализм**: нужно стараться избегать деталей, которые не нужны для выполнения задачи. Например: «можешь ещё картошки прихватить, но она тяжёлая, не хочешь тащить — не бери». Если картошка не нужна, её незачем упоминать.

Теперь вы знаете, как правильно составлять алгоритмы как для себя, так и для других.

Те же правила работают и в программировании. Грамотно составленный алгоритм будет понятен компьютеру и приведёт к нужным результатам, а алгоритм без соблюдения этих правил — нет.

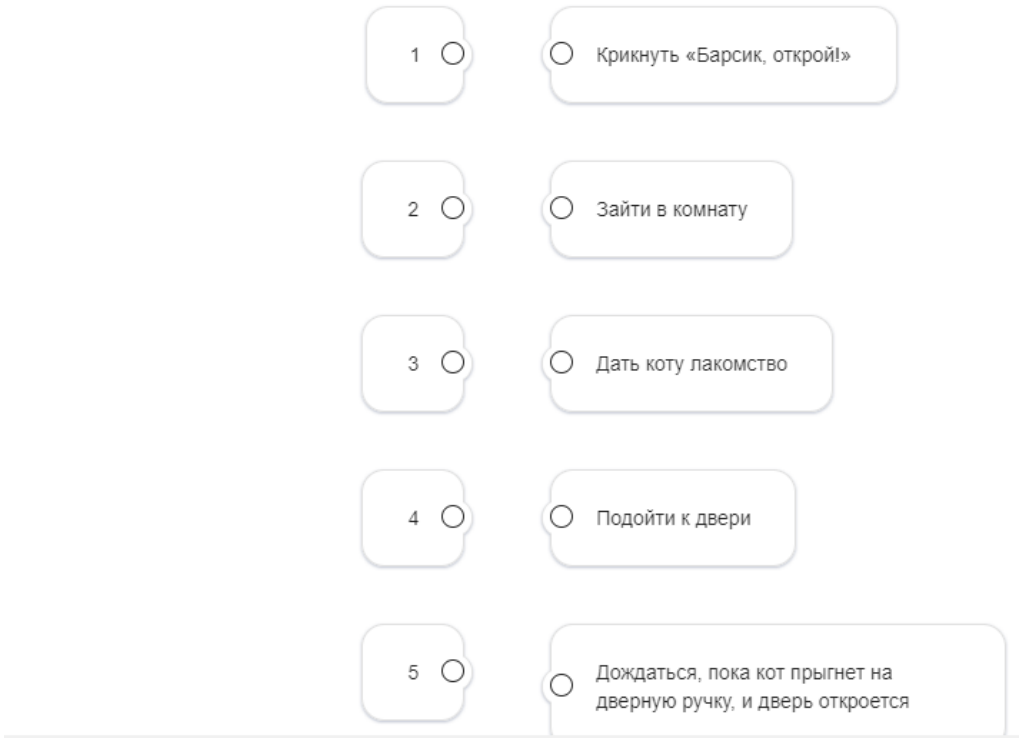
Чтобы научиться **программировать**, приучите себя думать о задачах как о последовательности шагов, которые нужно выполнить, чтобы достичь конкретной цели.

Вот алгоритм:

1. Берёте задачу.
2. Разбиваете на мелкие подзадачи.
3. Выполняете их в строгой последовательности.
4. Хвалите себя за работу.

Тренировка.

Переходим к программированию кота. Нужно натренировать Барсика открывать вам двери, когда вы несёте тяжёлые сумки с продуктами. Для этого соедините номера шагов с инструкциями.



Функции и переменные

Мы узнали, как алгоритмическое мышление работает во взаимодействии с другими людьми (и котами), а теперь давайте научимся видеть алгоритмы в вещах, которые нас окружают. Они повсюду: в чайниках, микроволновках, светофорах, компьютерах и стиральных машинах.

Например, чтобы вскипятить воду, чайник выполняет такой алгоритм:

1. Проверить, есть ли вода (если воды нет — не включаться);
2. Нагреть воду до 100 °C;
3. Выключиться, когда температура воды станет равна 100 °C.

Как видите, алгоритмы в технике подчиняются тем же правилам, что и в жизни:

- точность: что конкретно должно сделать устройство;
- последовательность: все шаги выполняются чётко один за другим;
- вариативность: что делать в разных ситуациях.

Но есть важный нюанс. Несмотря на то что компьютеры и устройства в последние несколько лет научились лучше распознавать, что человек от них хочет, они всё равно требуют от программистов использования некоторых обязательных элементов.

Эти обязательные элементы такие: **функции**, **переменные**, **условия** и **циклы**. Узнав про них больше, вы научитесь создавать алгоритмы, понятные компьютеру.

Типичный алгоритм состоит из действий, которые нужно выполнить. Например, алгоритм «Вскипятить» включает в себя такие действия:

```
вскипятить воду = проверить, есть ли вода
                  +
                  нагреть воду до 100 °C
                  +
                  если температура воды равна 100 °C,
                  выключиться
```

Как видите, внутри алгоритма описаны сразу несколько задач: «включись», «нагрей воду», «выключись». Всё это — функции, и внутри них есть код.

Функция в программировании — это список действий, направленных на решение одной конкретной задачи.

Выберем одну из функций — «Нагреть воду» — и осторожно заглянем внутрь. Там — код на Python:

```python
def boil(water_temp):
    while water_temp < 100:  # пока температура воды меньше 100°C,
        water_temp = water_temp + 10  # нагреваем воду на 10°C.
        # выводим текущую температуру.
        print('Текущая температура', water_temp, '°C')
    else:
        # выводим сообщение «Чайник закипел!».
        print('Чайник закипел!')
```

Вот как это работает: кто-то придумал функции уникальное имя (boil по-английски означает «нагреть») и поместил внутрь код, который поднимает температуру воды на 10 градусов до тех пор, пока чайник не закипит. Теперь каждый раз, когда вы будете давать чайнику команду «Нагрей воду», он будет выполнять код внутри функции boil.

Чаще всего в программировании происходит так: если вам нужно выполнить какое-то действие, вы либо используете существующую функцию, либо пишете свою, новую. Именно поэтому функций существует неописуемо много: сколько разнообразных действий, столько и функций.

Сразу научимся определять функции на глаз:

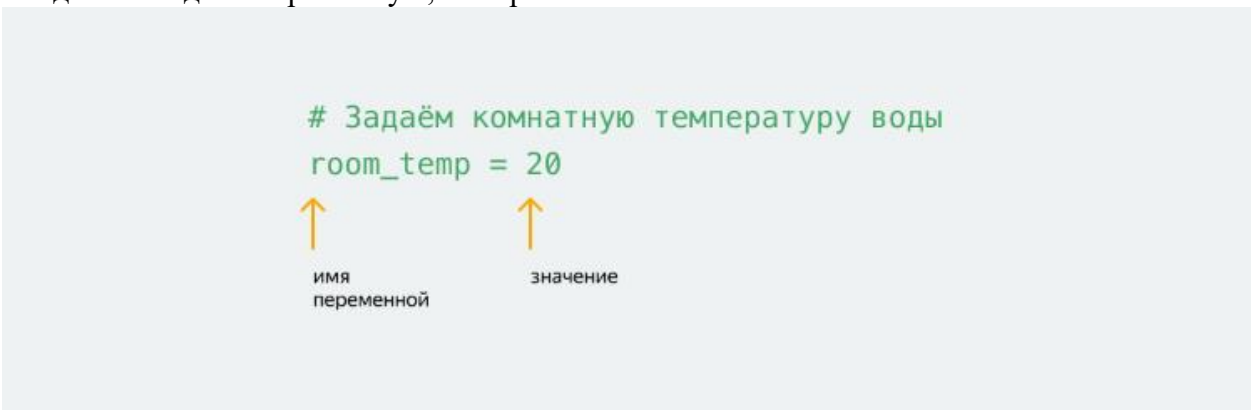
Любые действия, которые мы совершаем, происходят с каким-то объектом. В нашем примере с чайником такой объект — вода, а точнее — её температура water_temp. Это **аргумент функции** boil — то, что она принимает «на вход» и с чем будет совершать действия, которые мы опишем внутри функции.

Переменные

Любые действия, которые мы совершаем, нацелены на какой-то объект. В программировании такими объектами выступают данные: числа, названия и т.д. Чтобы не запутаться, данные в коде «хранят в подписанных коробках». Такие коробки для хранения данных называются переменные.

В примере с чайником объект, над которым совершается действие — это температура воды. Код обращается к переменной («заглядывает в коробку») и берет оттуда текущее значение температуры воды. Дальше он это значение увеличивает, пока оно не станет равно 100 градусам. Данные, с которыми работает функция, могут меняться в процессе, именно поэтому ящики для хранения и называются «переменные».

Когда мы создаём переменную, мы присваиваем ей значение.



Что произошло? Мы создали переменную с именем room_temp и поместили в неё число 20 — температуру воды. Дальше эта переменная будет использоваться в функции boil: её значение будет расти вместе с температурой воды в чайнике. Когда значение достигнет 100, чайник сообщит, что он закипел.

А теперь соберём код воедино и, наконец, вскипятим чайник.

Код PYTHON

```
room_temp = 20  # Задаём комнатную температуру воды.

def boil(water_temp):
    while water_temp < 100:  # Пока температура воды меньше 100°C,
        water_temp = water_temp + 10  # нагреваем воду на 10°C.
        # Выводим текущую температуру.
        print('Текущая температура', water_temp, '°C')
    else:
        # Выводим сообщение «Чайник закипел!».
        print('Чайник закипел!')

# А теперь запустим функцию.
boil(room_temp)
```

Запустить код

Результат

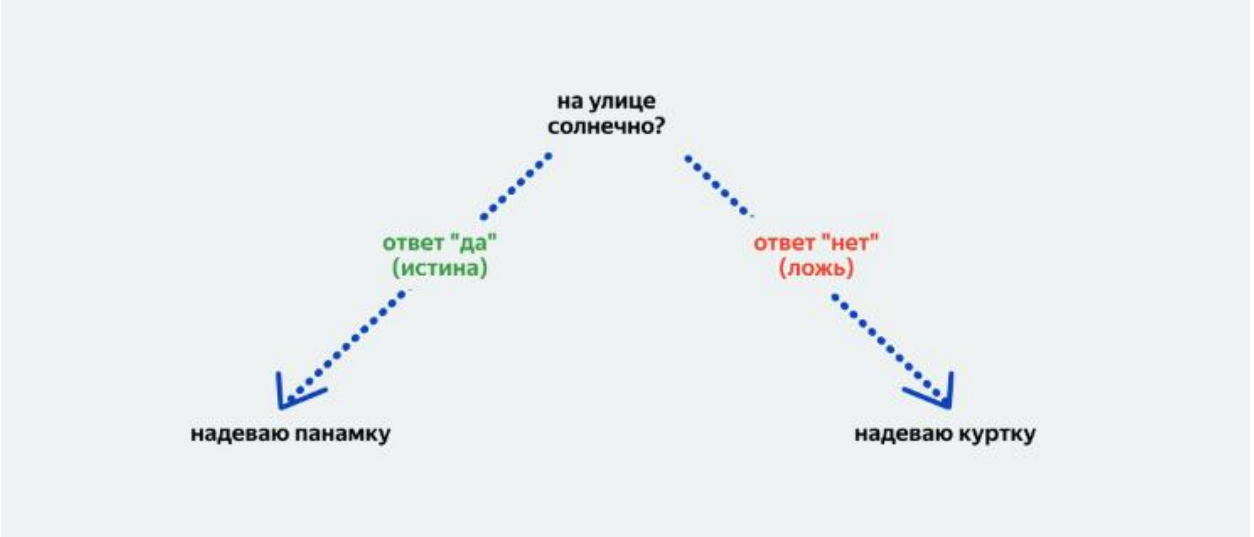
```
Текущая температура 30 °C
Текущая температура 40 °C
Текущая температура 50 °C
Текущая температура 60 °C
Текущая температура 70 °C
Текущая температура 80 °C
Текущая температура 90 °C
Текущая температура 100 °C
Чайник закипел!
```

Условия

Условие в алгоритмах — это своеобразная развилка, которая определяет следующие действия в зависимости от обстоятельств.

Представьте, что вы собираетесь погулять. Если в прогнозе погоды сказано, что будет солнечно, то вы решаете одеться легко и надеть панамку. Если нет — надеваете куртку.

«Если будет солнечно» — это условие. Если оно истинно, вы выполняете один пункт алгоритма, а если нет — другой.



Абсолютное большинство вещей, которые нас окружают, работают по тому же принципу:

- телевизор включится, если нажать кнопку на пульте;
- чайник выключится, если вода вскипела.

Условия помогают создавать алгоритмы, которые будут выбирать действия в зависимости от ситуации. Именно они позволяют уместить противоречивые желания в понятную последовательность шагов.

С одной стороны, вы хотите, чтобы ваш дом был как крепость: недоступный для всяких чужаков и непрошеных гостей. С другой — хотите, чтобы вы сами могли быстро и легко заходить в свою квартиру. Налицо противоречие, которое как раз под силу разрешить с помощью условий.

Если в дом пришёл чужак (у которого нет ключа, или он не знает пароль), надо его не пускать. Если хозяин (с ключом, паролем) — пустить.

Именно по такому принципу работает сигнализация.

Код PYTHON

```python
right_password = '1234'

def check_pass(password):
    if password == right_password:
        print('Пароль', password, '- верный. Сигнализация отключена.')
    else:
        print('Сработала сигнализация!')

# Попробуем ввести правильный пароль.
check_pass('1234')

# А теперь к нам пытается пробраться чужак.
check_pass('5432')
```

Запустить код

Результат

```
Пароль 1234 - верный. Сигнализация отключена.
Сработала сигнализация!
```

В программировании условия записываются с помощью ключевых слов, таких как if и else.

Слово if проверяет, истинно ли какое-то утверждение. Если утверждение, например пароль, верное, выполнится один блок команд (в нашем случае — сигнализация отключится).

Если утверждение ложно, алгоритм пропустит этот блок кода и перейдёт к другому — указанному после else. В нашем случае — сигнализация сработает.

Если условие — правда, выполняется код после if. Если условие — не правда, выполняется код после else.

Тренировка

Прочитайте код сигнализации и скажите, что произойдёт, если вы введёте вместо "start" код 379.

PYTHON

```python
def quiz(secret_num):
    if secret_num  == '279':
        print('Пароль 279 устарел. Попробуй другой.')
    if secret_num  == '379':
        print('Вход разрешен. Пароль 379 отключил сигнализацию и открыл двери.')
    else:
        print('Звоню в полицию! Включаю сирену! Попытка взлома!')

quiz('start')
```

- ◯ Сигнализация отключится и впустит вас
- ◯ Включится сирена, и будет вызван пост охраны
- ◯ Ничего не произойдёт

Циклы

Цикл в алгоритме — это способ повторять определённые действия или шаги несколько раз, пока выполняется определённое условие.

Давайте представим, что вы хотите приготовить яичницу. В холодильнике у вас лежат яйца, а вам нужно переложить их на стол. Чтобы сделать это, вы повторяете следующие действия: берёте одно яйцо, кладёте его на стол, затем берёте следующее яйцо и так далее, пока не перенесёте их все.

Конечно, вы можете написать алгоритм такого типа:

Но можно всё сделать проще с помощью циклов. В Python существует цикл while, который идеально подходит для нашей задачи, который идеально подходит для нашей задачи. Обратите внимание, что количество яиц указано просто для примера. На самом деле цикл будет выполняться, пока яйца не закончатся: яиц может быть десять, двадцать, сколько угодно.

Код PYTHON

```python
eggs = 10  # Переменная с количеством яиц в холодильнике.
while eggs > 0:  # Пока количество яиц в холодильнике больше нуля,
    print('Берём яйцо и кладём его на стол.')
    eggs = eggs - 1  # из переменной eggs отнимаем единицу,
    print('Яиц в холодильнике осталось: ', eggs)  # переносим на стол.
# А это выполнится, когда цикл закончится.
print('Все яйца на столе.')
```

Запустить код

Результат

```
Берём яйцо и кладём его на стол.
Яиц в холодильнике осталось:  9
Берём яйцо и кладём его на стол.
Яиц в холодильнике осталось:  8
Берём яйцо и кладём его на стол.
Яиц в холодильнике осталось:  7
Берём яйцо и кладём его на стол.
Яиц в холодильнике осталось:  6
Берём яйцо и кладём его на стол.
Яиц в холодильнике осталось:  5
Берём яйцо и кладём его на стол.
Яиц в холодильнике осталось:  4
Берём яйцо и кладём его на стол.
Яиц в холодильнике осталось:  3
Берём яйцо и кладём его на стол.
Яиц в холодильнике осталось:  2
Берём яйцо и кладём его на стол.
Яиц в холодильнике осталось:  1
Берём яйцо и кладём его на стол.
Яиц в холодильнике осталось:  0
Все яйца на столе.
```

Как видите, цикл while работает так: «пока условие, указанное после while, — правда, нужно выполнить следующий блок кода». Он будет выполнять его до момента, когда в переменной eggs останется ноль, а значит, утверждение while eggs > 0 станет ложью.

Циклы позволяют автоматизировать повторяющиеся задачи, делают код компактнее и сильно экономят время.

Функции

Функция — описание какого-то действия, которое создаётся, чтобы не повторять одно и то же много раз. Например, если вы с коллегой всегда обедаете в одном месте в 13:00, достаточно просто сказать ему «пойдём обедать», чтобы он пришёл в нужное кафе в нужное время.

Чтобы вы привыкали к сленгу программистов, сразу скажем, что «объявить функцию» — значит впервые описать действие, а «вызвать функцию» — обратиться к уже описанному действию.

Если вспомнить историю с чайником, вот так объявляли функцию boil():

```python
room_temp = 20  # Задаём комнатную температуру воды.

def boil(water_temp):
    while water_temp < 100:  # Пока температура воды меньше 100°C,
        water_temp = water_temp + 10  # нагреваем воду на 10°C.
        # Выводим текущую температуру.
        print('Текущая температура', water_temp, '°C')
    else:
        # Выводим сообщение «Чайник закипел!».
        print('Чайник закипел!')
```

Если коротко:

- функция отвечает за действие,
- у каждой функции есть имя,
- перед созданием новой функции нужно указывать слово def,
- когда вызываем функцию — в скобочках указываем аргументы.

Аргументы — те входные данные, которые нужны функции для работы. Например, указания, к кому или к чему эту функцию нужно применить, или значения определённых переменных. Их указывают в скобках после названия функции. Например, «пойдём обедать (Олег)» даёт функции «пойдём обедать» информацию, что вы хотите пообедать с конкретным человеком. В этом случае Олег — аргумент.

• В функции boil() вы передавали переменную room_temp (температура воды), а на выходе получали оповещение о том, что вода вскипела, или другое значение температуры после нагрева. В этом случае room_temp — аргумент.

Тренировка

В функции say_hi есть один аргумент — переменная name.

```python
def say_hi(name):
    print('Приветик,' + name + '! Как дела?')
```

Задача этой функции — быть вежливой, то есть здороваться со всеми, с кем мы попросим. При запуске функция say_hi подставляет значение переменной name после слова «Приветик» и перед фразой «Как дела?». Важно оговориться, что функция не обязательно будет здороваться по имени. Если вы присвоите переменной значение 123robot15, она обратиться с приветствием именно к 123robot15.

Давайте сделаем так, чтобы функция поздоровалась с нами!

Напишите своё имя в качестве аргумента для функции say_hi в 4 строчку кода

Код

```python
def say_hi(name):
    print('Приветик,' + name + '! Как дела?')

say_hi('') # напишите имя здесь между кавычками ''
```

Главное, что нам нужно запомнить:

- Аргументы — это данные, которые мы предоставляем функции, чтобы она могла выполнить свою задачу.
- Передавать можно что угодно — числа, слова, значения переменных и так далее.

ПРОГРАММИРОВАНИЕ

Напишем свой собственный таймер-будильник, который будет засекать время на обучение и присылать вам сигнал, когда можно будет включать сериал.

Сначала пропишем алгоритм действий:

1. Создать переменную, которая будет хранить количество времени, отведённого на обучение.
2. Создать функцию, которая будет отсчитывать время, заданное в переменной.
3. Когда время в переменной станет равно нулю, вывести сообщение «Пора смотреть сериал!».

PYTHON

```
# Для работы с таймером его сначала надо импортировать в программу.
from time import sleep

timer_seconds = 10

def my_timer():
    # Строчка ниже говорит таймеру, сколько секунд ждать.
    sleep(timer_seconds)
    # Как только этот таймер отсчитает время, выводим сообщение.
    print('Пора смотреть сериал!')
```

Вопрос 1: Как называется функция?

- ◯ `import time`
- ◯ `timer_seconds`
- ◯ `my_timer()`

Вопрос 2: Что может быть аргументом функции?

- ☐ `sleep`
- ☐ 10
- ☐ `timer_seconds`

Закрепление

1. Какими словами вы расскажете своей бабушке, что значит мыслить алгоритмически?

- ◯ Уметь разбивать сложные задачи на более простые шаги и последовательно выполнять их, чтобы достичь результата.
- ◯ Уметь переводить сложные задачи на язык компьютера, чтобы помочь ему достичь результата.
- ◯ Уметь использовать для решения сложных задач цикл, условие и переменные.

2. Программист Олег прочитал утром на холодильнике записку:

«Купи шоколадку
Если будут бананы
Купи четыре»

После чего он по традиции перевел ее в своей голове в алгоритм на «питоне».

PYTHON

```
def go_shopping():
    # Переменные в функции.
    bananas = 2
    chocolate = 0
    # Если бананы есть в наличии.
    if bananas > 0:
        chocolate = chocolate + 4
        print('Купил шоколадки. Количество:', chocolate )
    else:
        chocolate = chocolate + 1
        print('Купил шоколадки. Количество:', chocolate )

go_shopping()
```

Вопрос: исходя из алгоритма, купит ли Олег в итоге бананы и если да, то сколько? Задача на внимательность и алгоритмическое мышление.

- ◯ Купит 2 штуки
- ◯ Купит 4 штуки
- ◯ Не купит

3. Программистка Катерина научила свой светильник включаться и выключаться по расписанию. Выберите значение переменной time_hour, при которой ночник не будет светить

PYTHON

```
# Сообщаем функции текущее время.
def night_light(time_hour):
    # Условия при которых будет включаться и выключаться ночник.
    if time_hour < 7 or time_hour > 20:
        print('Ночник светит')
    else:
        print('Ночник выключен')

night_light(7)
```

- ◯ 0
- ◯ 4
- ◯ 6
- ◯ 16

4. Что-то пошло не так и Катя с Олегом ходят вечно потные, потому что их кондиционер держит температуру в 40 градусов, когда дома и так жара. Как исправить код, чтобы решить проблему?

PYTHON

```
# Указываем желаемую нам температуру.
needed_temp = 40

def conditioner(temp):
    # Запускаем цикл, пока температура меньше желаемой.
    while temp < needed_temp:
        # Увеличиваем значение температуры на 1 градус.
        temp = temp + 1
        print('Нагрел воздух до', temp, '°C')
    # запускаем цикл пока temp больше желаемой
    while temp > needed_temp:
        # Уменьшаем значение температуры на 1 градус.
        temp = temp - 1
        print('Охладил воздух до', temp, '°C')

# Запускаем программу кондиционера.
conditioner(20);
# Тестируем работу кондиционера и ищем неисправность в коде.
```

- ◯ Нужно поменять в цикле температуру, на которую нагревается/охлаждается воздух с 1 °C до 3 °C
- ◯ Нужно задать другое значение переменной `needed_temp`
- ◯ Нужно переписать код целиком, потому что он изначально содержит неправильную логику

5. Олег мечтал, чтобы рано или поздно ему удалось запрограммировать кота, а пока он записал его поведение в формате алгоритма.

```python
food = 0
# Пока еда не получена, кот совершает следующие действия.
while food < 1:
    print('Ляг на стол')
    print('Пройдись по клавиатуре')
    print('Скинь кружку со стола')
    print('Подточи когти о диван')
    food = food + 1
```

Сколько раз коту необходимо повторить цикл действий, чтобы у него появилась еда?

- ◯ Один
- ◯ Два
- ◯ Неизвестно