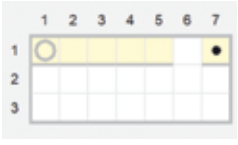
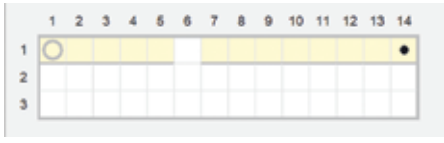
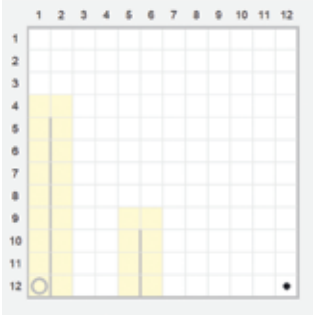
5 На поле *Робота* размещён «забор» — горизонтальная стена. Забор нужно «покрасить» — написать программу для закрашивания всех клеток сверху стены (задача `upr_14_5`). В «заборе» могут быть одни «ворота» — клетка без линий. Длина «забора» и расположение «ворот» неизвестны.





6* Решите задачу `upr_14_6`. Используйте вспомогательный алгоритм для обхода каждой из двух стен, расположенных на поле *Робота*.



§ 15. Алгоритмическая конструкция *ветвление*

15.1. Команда *ветвление*

Довольно часто на поставленный вопрос человек получает ответ «да» или «нет». В зависимости от ответа человек определяет свои действия и выполняет одну или другую «команду» или группу «команд» (пример 15.1).

Роботы и другие технические устройства также могут выпол-

Пример 15.1. Выбор обуви весной в зависимости от погоды.

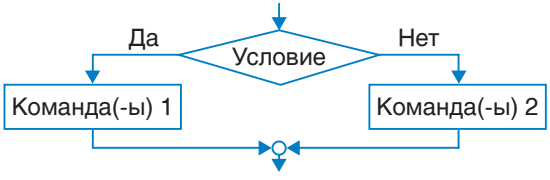
```
Если на улице дождь, то
    надеть резиновые сапоги
Иначе
    надеть туфли
```

В данном примере в текущий момент времени может быть выполнена только одна команда из двух: или надеть сапоги, или надеть туфли.

нять различные действия в зависимости от условий. Если условие истинно (на вопрос получен ответ «да»), то выполняется один набор действий, если ложно, то другой.

Алгоритмическая конструкция *ветвление* обеспечивает выполнение одной или другой последовательности команд в зависимости от истинности или ложности некоторого условия.

Команда *ветвление* может изображаться на блок-схеме следующим образом:



Принцип работы команды *ветвление* описан в примере 15.2.

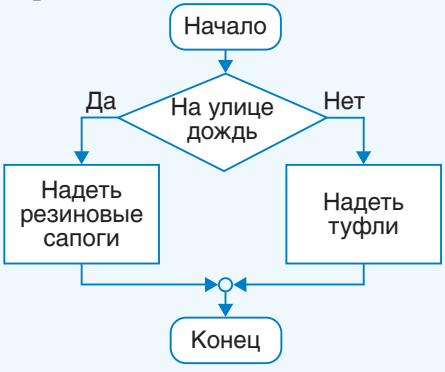
В языке программирования Python для записи конструкции *ветвление* используется команда `if`. Формат записи команды:

```
if <условие>:
    команда(-ы) 1
else:
    команда(-ы) 2
```

Строка `if <условие>:` является **заголовком ветвления**. Эту строку можно прочитать так:

Пример 15.1. *Продолжение.*

Блок-схема данного алгоритма будет выглядеть следующим образом:



Понятие ветвления используется в различных сферах человеческой деятельности.

В ботанике под ветвлением побегов понимают процесс увеличения числа побегов у растений.

Ветвления используются в дорожной разметке и картографии.

При употреблении термина в переносном смысле под ветвлением понимают наличие нескольких путей, направлений, сюжетных линий и т. д.

Пример 15.2. Принцип работы команды *ветвление*.

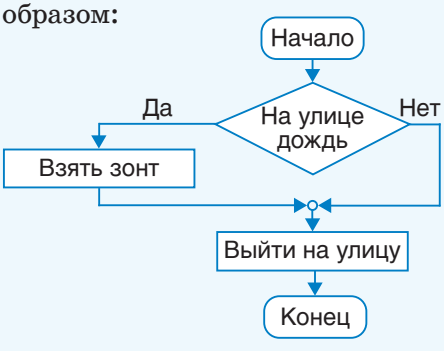
Выполнение конструкции начинается с проверки условия. Если условие истинно, то выполняется последовательность команд 1, если условие ложно, то выполняется последовательность команд 2. При такой организации алгоритма может выполниться **только одна** из двух последовательностей команд. Другая последовательность команд будет проигнорирована.

Пример 15.3. Выход на улицу осенью.

```
Если на улице дождь, то
    взять зонт
выйти на улицу
```

В данном примере используется сокращённая форма команды *ветвление*. Если условие истинно, то выполняется команда *взять зонт*. Если условие ложно, то никаких действий не происходит. Команда *выйти на улицу* выполняется всегда, независимо от истинности или ложности условия.

Блок-схема данного алгоритма будет выглядеть следующим образом:

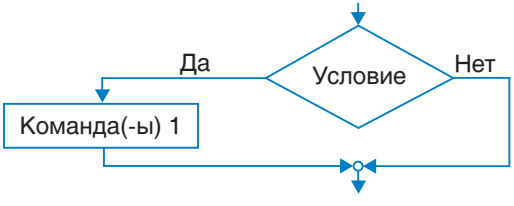


«Если условие верно, то...». В следующей строке со сдвигом записывается последовательность команд 1. После слова `else` в следующей строке со сдвигом записывается последовательность команд 2. Обратите внимание на то, что символ «:» ставится как в конце заголовка ветвления, так и после слова `else`.

Команда *ветвление* может быть записана в **полной** или **сокращённой** формах.

Полная форма команды *ветвления* организует выполнение двух разных наборов команд, из которых выполняется только один. В сокращённой форме один из наборов команд отсутствует. Если отсутствует набор команд, соответствующих ситуации, когда условие ложно, то никакие действия не выполняются (пример 15.3).

Блок-схема сокращённой формы *ветвление*:



На языке программирования Python команда записывается следующим образом:

```
if <условие>:
    команда(-ы) 1
```

В качестве последовательности команда(-ы) 1 и/или 2 может выступать другая команда ветвления. То есть, если истинно (ложно) одно условие, то может потребоваться проверить другое условие. Такие конструкции образуют **вложенные ветвления** (пример 15.4).

В языке Python для записи вложенных ветвлений (в случае ложности первого условия) можно применять конструкцию `elif`. Запись конструкции:

```
if <условие1>:
    команды 1
elif <условие2>:
    команды 2
else:
    команды 3
```

Ключевое слово `elif` в записи команды по существу заменяет два других: `else` и `if`. Блоков `elif` может быть несколько в том случае, если необходимо проверить не два условия, а больше.

15.2. Составные условия

В качестве условия в алгоритмах с циклами и ветвлениями используется любое понятное исполнителю этого алгоритма

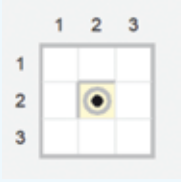
Пример 15.4. Имеется три монеты, среди которых одна фальшивая. Известно, что фальшивая монета легче настоящих монет. Как найти фальшивую монету за минимальное число взвешиваний на чашечных весах без гирь? За какое количество взвешиваний можно определить фальшивую монету?

Представим словесное описание алгоритма решения этой задачи.

```
Положить на каждую чашу весов по одной монете (монета 1
и монета 2)
Если весы в равновесии, то
    фальшивая монета 3
Иначе
    Если монета 1 тяжелее, то
        фальшивая монета 2
    Иначе
        фальшивая монета 1
```

За одно взвешивание мы можем определить фальшивую монету.

Пример 15.5. Начальная обстановка поля *Робота*.



Первое условие состоит из двух простых: *A* = `wall_left()` и *B* = `cell_is_filled()`. Условие может быть записано как *A* **И** *B*. Это условие верно только тогда, когда верны и *A*, и *B*. Условие *A* = 1 (истинно), условие *B* = 1 (истинно), условие *A* **И** *B* = 1 (истинно).

Второе условие может быть записано как *A* **ИЛИ** *B*, где *A* = `wall_up()`, *B* = `wall_down()`. Условие *A* = 1, условие *B* = 0. Значит, условие *A* **ИЛИ** *B* = 1 (истинно).

В третьем условии операция `not` отрицает составное условие `wall_right() or not wall_up()`. Условие может быть записано как **НЕ (***A* **ИЛИ НЕ** *B***)**. Соблюдая порядок выполнения логических операций, определим истинность или ложность данного условия:

1) **НЕ** *B* = 0, так как *B* = 1;

2) *A* **ИЛИ НЕ** *B* = 0, так как *A* = 0;

3) **НЕ (***A* **ИЛИ НЕ** *B***)** = 1.

Для других обстановок *Робота* истинность или ложность приведённых составных условий может быть другой.

высказывание, которое может быть либо истинным, либо ложным.

Все условия, с которыми нам приходилось до сих пор встречаться при составлении алгоритмов для *Робота*, были простыми высказываниями. Для исполнителя *Робот* можно строить и составные условия. Составное условие для *Робота*, так же как и для любого другого исполнителя, образуется из нескольких простых условий, соединённых друг с другом логическими операциями.

С логическими операциями над высказываниями вы уже знакомы. В Python используют следующие логические операции:

Логическая операция	Запись в Python
НЕ	`not`
И	`and`
ИЛИ	`or`

Истинность или ложность составного условия для исполнителя *Робот* определяется для конкретной обстановки.

Пример 15.5. Проверить для *Робота* следующие составные условия:

1. `wall_left() and cell_is_filled()`
2. `wall_up() or wall_down()`
3. `not (wall_right() or not wall_up())`

15.3. Использование команды *ветвление* для исполнителя *Робот*

Пример 15.6. *Робот* должен закрасить клетку, которая находится за стеной. В зависимости от обстановки обход стены может осуществляться по-разному.

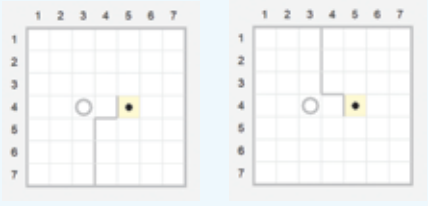
В начале *Робот* должен сдвинуться вправо. Если стена снизу, то сверху свободно и можно обойти стену сверху, в противном случае *Робот* обходит стену снизу. После обхода стены *Робот* закрашивает клетку. Алгоритм можно записать следующим образом:

```
Вправо
Если стена снизу, то
    Вверх
    Вправо
    Вниз
Иначе
    Вниз
    Вправо
    Вверх
Закрасить
```

Для решения этой задачи использована полная форма команды *ветвление*.

Пример 15.7. *Робот* находится на неизвестной клетке поля без линий. Он должен закрасить клетку слева от себя.

Пример 15.6. Две разные обстановки поля *Робота*.



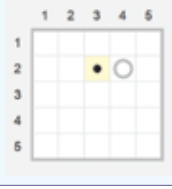
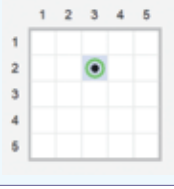
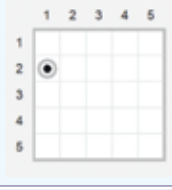
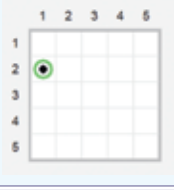
Программа для исполнителя *Робот*:

```
from pyrob.api import *

@task
def prim_15_6():
    move_right()
    if wall_down():
        move_up()
        move_right()
        move_down()
    else:
        move_down()
        move_right()
        move_up()
    fill_cell()

run_tasks()
```

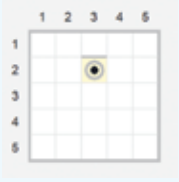
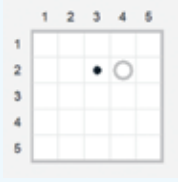
Пример 15.7. Программа для исполнителя *Робот*.

```
from pyrob.api import *

@task
def prim_15_7():
    if not wall_left():
        move_left()
        fill_cell()

run_tasks()
```

Пример 15.7. *Продолжение.*
Результат работы программы для разных начальных обстановок:

Начальная обстановка	Результат
	
	

Пример 15.8. Две разные обстановки поля *Робота*.

Программа для исполнителя *Робот*:

```
from pyrob.api import *
@task
def prim_15_8():
    if wall_up():
        fill_cell()
    elif wall_down():
        fill_cell()
    else:
        move_left()
run_tasks()
```

Робот может переместиться влево для закрашивания клетки только в том случае, если слева нет стены, иначе он не выполнит никаких действий и останется на месте. Поэтому, прежде чем сдвинуться влево, *Робот* должен проверить, свободно ли слева. В том случае, когда *Робот* находится возле левой границы поля, он ничего не делает, оставаясь на своём месте.

Результат работы данной программы зависит от начального положения *Робота*. Для решения этой задачи использована сокращённая форма команды *ветвление*.

Пример 15.8. *Робот* находится на произвольной, не прилежащей к границе, клетке поля. Вокруг клетки сверху или снизу могут быть стены. Он должен закрасить клетку, на которой находится, если у неё есть хоть одна стена, иначе *Робот* должен сдвинуться влево.

Программа для решения этой задачи может быть записана двумя способами:

1) с использованием конструкции `elif`;

2) с использованием составного условия.

В первом случае *Робот* проверяет условие «стена сверху» и в случае его ложности переходит на проверку условия «стена снизу». Во втором случае используется составное условие: «стена сверху ИЛИ стена снизу».

Пример 15.8. *Продолжение.*

Исполняемый фрагмент программы для второго способа:

```
if wall_up() or wall_down():
    fill_cell()
else:
    move_left()
```

1. Что такое алгоритмическая конструкция *ветвление*?
2. Из чего состоит заголовок команды *ветвление*?
3. Чем отличается форма записи полной конструкции *ветвление* от сокращённой?
4. Какая конструкция может использоваться для записи вложенных ветвлений?
5. Что такое составное условие?
6. Какие логические операции можно использовать для записи составных условий?
7. Как определить истинность составного условия?

Упражнения

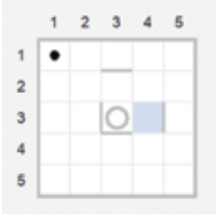
1 Выделите алгоритмическую конструкцию *ветвление* в отрывке из поэмы А. С. Пушкина «Руслан и Людмила» и изобразите эту конструкцию с помощью блок-схемы.

У Лукоморья дуб зелёный;
Златая цепь на дубе том:
И днём и ночью кот учёный
Всё ходит по цепи кругом;
Идёт направо — песнь заводит,
Налево — сказку говорит.
Там чудеса: там леший бродит,
Русалка на ветвях сидит...[1]

[1] Пушкин А. С. Руслан и Людмила : поэма. М., 1996. С. 5.

2 Определите, какие из составных условий истинны, а какие ложны для заданной обстановки поля *Робота*.

Начальная обстановка	Условия

	`wall_left() or cell_is_filled()`
	`wall_up() and wall_down()`
	`not cell_is_filled() and not wall_right()`
	`not (wall_up() or not wall_right())`
	`wall_down() and cell_is_filled()`
	`(wall_up() or wall_down()) and not wall_right()`

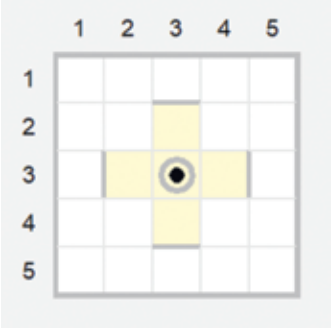
3 В каждом из условий упражнения 2 заменили операцию `and` на `or`, а операцию `or` на `and`. Изменится ли истинность приведённых условий?

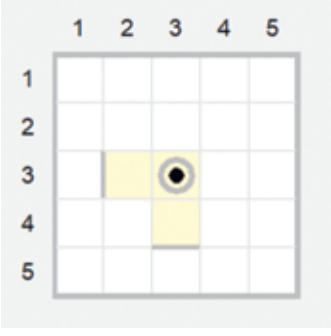
4 Для каждого из ложных условий упражнения 2 придумайте обстановку поля *Робота*, в которой данное условие будет истинным, а для каждого истинного — обстановку, в которой условие будет ложным.

5 Решите задачи `upr_15_5_1` и `upr_15_5_2`. Эти задачи аналогичны примеру 15.7 на с. 111. В задаче `upr_15_5_1` *Робот* должен закрасить клетку справа, а в задаче `upr_15_5_2` — снизу.

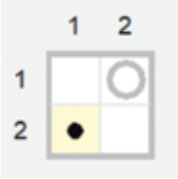
6* Решите задачи `upr_15_6_1` и `upr_15_6_2`.

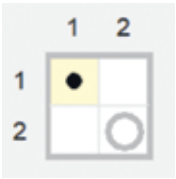
1. Клетка, на которой находится *Робот*, должна быть закрашена. Остальные клетки закрашиваются только тогда, когда есть стена с соответствующей стороны.





2. *Робот* находится в какой-то клетке поля размером 2 × 2. Он должен закрасить клетку в углу, противоположном начальному.





§ 16. Использование основных алгоритмических конструкций для исполнителя *Робот*

Последовательное выполнение команд в программе определяется структурой *следование*. Для организации повторяющихся действий в алгоритме используется команда *цикл*. Команда *ветвление* позволяет выполнять одну или другую последовательность команд в зависимости от истинности условия.

Следование, *цикл* и *ветвление* — **базовые алгоритмические конструкции.** Используя эти конструкции как элементы некоего «конструктора», можно составлять и разрабатывать любые алгоритмы.

Команды *цикл* и *ветвление* управляют порядком выполнения других команд в программе и относятся к командам управления. Использование алгоритмической

Перед человеком постоянно возникают разнообразные задачи, для которых существуют различные алгоритмы решения. При всём многообразии алгоритмов для их записи достаточно трёх алгоритмических конструкций (структур): *следование*, *цикл*, *ветвление*.

Это положение было выдвинуто в середине 1970-х гг. учёным Эдсгером Вибе Дейкстрой (1930–2002 гг.). Его труды оказали влияние на развитие информатики и информационных технологий. Дейкстра являлся одним из разработчиков концепции структурного программирования, участвовал в разработке языка программирования Алгол. Известен своими разработками в области математической логики и теории графов.