GeekBrains

Знакомство с Python

Часть 2

Познакомимся с циклами и функциями, используемыми в Python

Циклы

Типы циклов

Циклы призваны поддерживать принцип разработки DRY (Don’t Repeat Yourself — «Не повторяйся»).

Существует два типа циклов:

- for — используется, когда количество итераций (шагов) цикла известно,
- while — может использоваться, когда количество итераций (шагов) неизвестно.

Единственный способ создать бесконечный цикл — while. Этот цикл также удобно использовать, если заранее неизвестно, когда произойдёт событие, после которого можно продолжать работу.

Схемы циклов:

```
for <счетчик > in <последовательность>:
    тело цикла
```

```
while <условие>:
    тело цикла
```

Структура цикла for

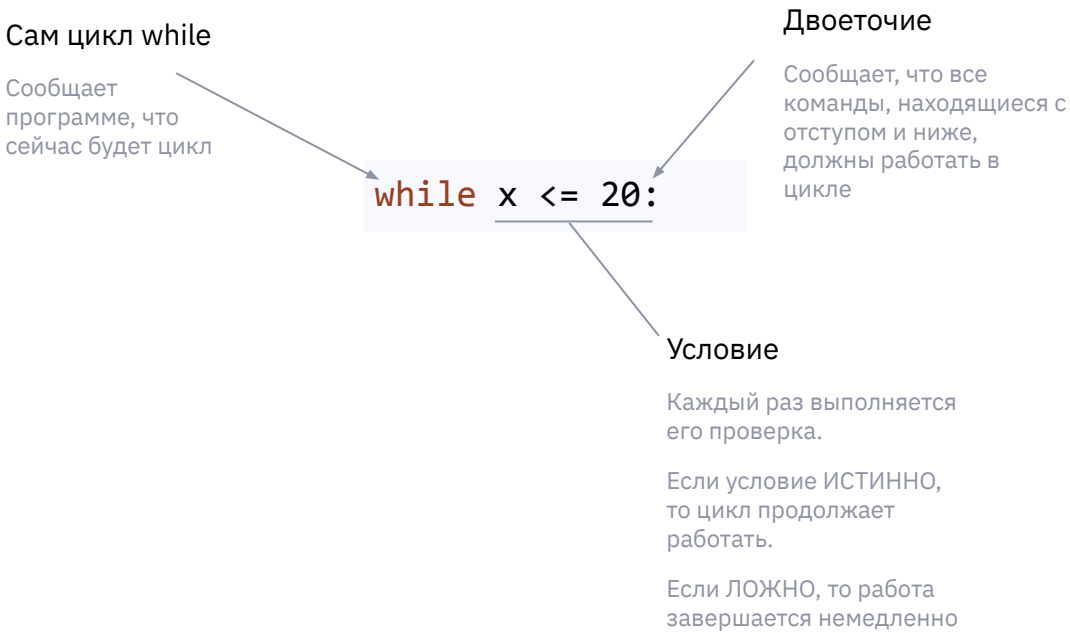
Подробнее про последовательности

range(n) — функция, которая позволяет генерировать последовательность чисел от 0 до указанного n. На каждой итерации цикла переменная-счётчик принимает следующее значение из последовательности, пока не будет достигнут конец.

Вместо range можно подставлять последовательности существующие в Python (списки, словари, множества). Тогда мы сможем обходить все элементы последовательности. Например, возьмём список чисел и каждое из них возведём в квадрат:

```
1 numbers = [1, 2, 3, 4, 5]
2
3 for i in numbers:
4     print(i**2)
```

Структура цикла while



Программа без определённого конца

Для лучшего понимания необходимости использования цикла while давайте разберём вот такой пример:

```
1 name = input('Как тебя зовут? ')
2 while name ≠ 'выход':
3     print(f'Привет, {name}!')
4     name = input('Введите еще имя или "выход", чтобы выйти ')
```

В данном коде пользователь может до бесконечности вводить своё имя и программа будет с ним здороваться. Момент, когда пользователю надоест и он введёт слово «выход», неизвестен никому. Поэтому цикл while здесь будет единственным решением.

Бесконечный цикл

Примеров использования цикла while много. Прямо сейчас у вас на компьютере работает как минимум несколько сотен циклов while, причём бесконечных.

Бесконечный цикл обеспечивает жизненный цикл любой программы, а выход осуществляется по специальному условию с помощью оператора **break**.

Перепишем предыдущую программу с использованием бесконечного цикла:

```
1 name = input('Как тебя зовут? ')
2 while True:
3     if name == "выход":
4         break
5     print(f'Привет, {name}!')
6     name = input('Введите еще имя или "выход", чтобы выйти ')
```

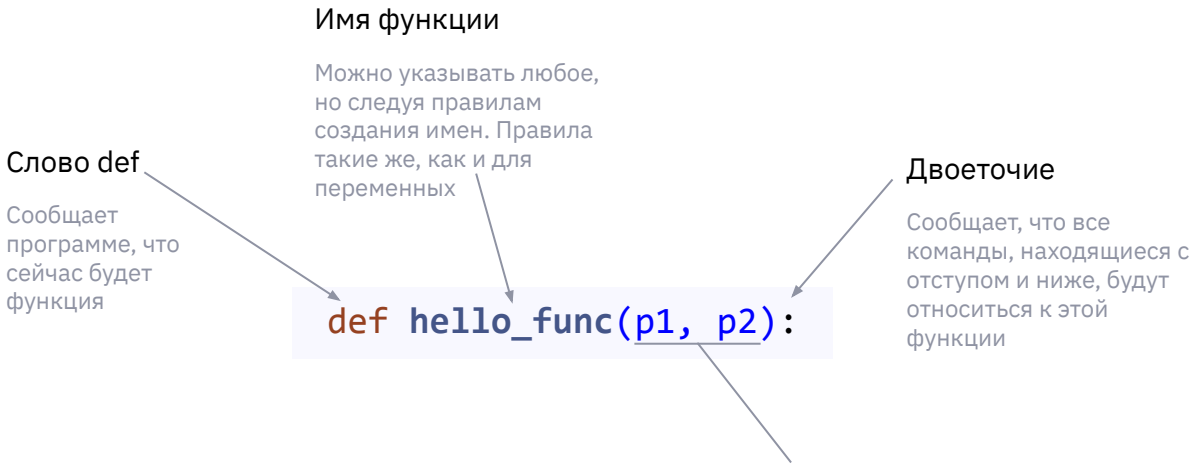
Функции

Создание функций

Функция — ещё один элемент, призванный следовать правилу DRY. Это фрагмент кода, выполняющий ту или иную задачу. Это также один из способов повторного использования кода, ведь одну и ту же функцию можно вызывать в своих программах снова и снова. Шаблон функции выглядит следующим образом:

```
def <имя_функции> (<параметры функции>):
    тело функции
```

Структура функции



Работа функции

Функция сама по себе работать не будет — её необходимо вызвать. Для этого в коде ниже просто вызываем её по имени:

```
1 def circle_square(radius):
2     result = 2*3.14*radius
3     print(result)
4
5
6 circle_square(20)          # Вызов функции
```

Параметры и возврат значений

Одна из особенностей функции — возможность принимать разные значения и выдавать разный результат. В примере выше мы можем вызвать функцию, чтобы рассчитать площадь любого круга, передав его радиус. Таких параметров может быть сколь угодно.

Но хорошие функции не выводят ничего на экран. Они вообще никак не вмешиваются в основной ход программы. Чтобы это стало возможным, полученное значение в ходе работы функции необходимо вернуть. Улучшенный вариант будет выглядеть так:

```
1 def circle_square(radius):
2     return 2 * 3.14 * radius    # Возвращаем значение
3
4
5 result = circle_square(20)      # Сохраняем значение в переменную
6 print(result)                   # выводим результат работы функции
```

Именованные параметры

Когда функция принимает в себя несколько параметров, то при её вызове мы передаем их в том же порядке, в котором они объявлены. Если необходимо изменить порядок, можно явно указать, что и в какой параметр мы записываем:

```
1 def expression(a, b, c):
2     return a + b * c
3
4
5 result = expression(c=5, b=3, a=4)
6 print(result)
```

Как думаете, какой результат получим?

Параметры по умолчанию

Иногда можно сделать так, чтобы функция работала при указании разного количества параметров. Для этого при объявлении функции можно определить значения по умолчанию:

```
1 def expression(a=1, b=1, c=1):
2     return a+b*c
3
4 result = expression(b=10)
```

Здесь можно не указывать ни одного параметра при вызове функции: для них определены значения по умолчанию. Но если мы укажем (как в нашем случае для b), то мы переопределим значение по умолчанию (1) нашим указанным значением (10).

Переменное количество параметров

Функции в Python умеют принимать разное количество параметров. Для этого существуют *args (аргументы) и **kwargs (именованные аргументы). Например, напишем функцию, которая считает среднее арифметическое любого количества оценок у ученика:

```python
def print_scores(student, *args):
    sum = 0
    for score in args:
        sum += score
    mid_score = sum/len(args)
    print(f"Студент: {student}, Средняя оценка: {mid_score}")

print_scores("Ivan", 5,4,3,5,3,4)
```

Здесь на первом месте указываем имя ученика, а дальше передаём любое количество чисел через запятую. Оператор * распакует содержимое данного параметра и позволит использовать как итерируемый объект.

Переменное количество параметров

Подобным образом работают и **kwargs. Только если *args был похож на список, то **kwargs больше похож на словарь.

```
def print_pet_names(owner, **pets):
    print(f"Владелец: {owner}")
    for pet, name in pets.items():
        print(f"{pet}: {name}")


print_pet_names("Кирилл", dog="Рекс", fish=["Дори", "Немо",
"Марлин"], turtle="Бульк")
```

Через запятую передаём произвольное количество пар ключ-значение и потом работаем с ними в функции.

Практическое задание (1/2)

Парадокс Монти Холла — одна из известных задач теории вероятностей, решение которой, на первый взгляд, противоречит здравому смыслу. Парадокс основан на американском телешоу «Let's Make a Deal» и назван в честь ведущего этой передачи.

Правила игры

Участнику игры нужно выбрать одну из трёх дверей. За одной из дверей находится приз, за двумя другими дверями ничего нет. Участник выбирает одну из дверей, например, номер 1, после этого ведущий, который знает, где находится приз, открывает одну из оставшихся дверей, например, номер 3, за которой ничего нет. После этого он спрашивает участника, не желает ли он изменить свой выбор и выбрать дверь номер 2? Участник может согласиться либо остаться при своём выборе.

Вопрос

Увеличиваются ли шансы участника выиграть приз, если он примет предложение ведущего и изменит свой выбор?

Предположения:

- Математики утверждают, что увеличатся на 60%.
- Разум подсказывает, что нет.

Практическое задание (2/2)

Задание

Напишите программу, которая подтвердит или опровергнет доводы математиков. Чтобы это выполнить напишите функцию которая принимает два параметра: номер двери, которую открываем при первом вопросе, и переменную булевого типа, которая означает, меняет ли человек свой выбор при второй попытке открытия двери. Функция должна возвращать переменную булевого типа, которая означает, выиграл человек или проиграл. Запустите функцию в цикле > 1000 раз и подсчитайте соотношение выигрышей к проигрышу при смене двери и при отказе от смены двери.

Помощь

Для решения задачи вам потребуется импортировать модуль random:

```
import random
```

и использовать его метод randrange. Метод randrange принимает в качестве аргумента два числа n1 и n2 и генерирует случайное число в диапазоне от n1 до n2-1:

```
random.randrange(2, 5)
# вернет либо 2, либо 3, либо 4
```

Вопросы?

Вопросы?

Вопросы?