ТЕХНОЛОГИЧЕСКАЯ КАРТА ЗАНЯТИЯ

Тема занятия: Разработка собственных функций.

Аннотация к занятию: на данном уроке обучающиеся знакомятся с функциями в Python. В первой части урока они учатся определять функцию и применять её. Во второй части урока они изучают область видимости переменных в функциях.

Цель занятия: знакомство учащихся с основами создания функций в Python.

Задачи занятия:

- познакомить с понятием «функция в Python»;
- научить создавать функции;
- изучить особенности области видимости переменной внутри функции;
- сформулировать определение локальной и глобальной переменной;
- познакомить с понятием лямбда-функции.

Ход занятия

Этап занятия	Время	Деятельность педагога	Комментарии, рекомендации для педагогов
Организационный этап	2 мин.	Здравствуйте! Очень рада вас видеть на уроке. Мы почти завершили большой блок о программировании на Python	Приветствие. Создание в классе атмосферы психологического комфорта
Постановка цели и задач занятия. Мотивация учебной деятельности обучающихся	10 мин.	Мы уже умеем работать с данными: загружать, обрабатывать и сохранять их. Однако, все программы, которые мы написали, можно назвать одноразовыми. Они работают только сами для себя. Вы не сможете использовать такой код повторно. Это не очень удобно. **Вопрос для обсуждения** Представьте, что вы разработали четыре модели машинного обучения и захотели объединить их в супермодель. Как это сделать? **Возможный ответ обучающихся:** скопировать код всех программ и вставить в один файл. Тогда придётся долго его форматировать и проверять логику кода. Непрактично.	Способствовать обсуждению мотивационных вопросов

		Для такой задачи создают функции — с ними мы сегодня и познакомимся.	
Изучение нового материала	50 мин.	Создание функции называют определением. Мы сообщаем компилятору, что теперь за ключевым именем скрывается определённый код. Создадим функцию, которая выведет знакомую строку «Привет, мир». Для этого пропишем ключевое слово def, а после — имя будущей функции. Имя может быть любым, как у переменных. Главное — отражать смысл функции. После имени нужно поставить двоеточие: теперь все строки с отступом будут принадлежать функции. Их называют телом функции. `[1] def hello():` `  print("Hello word")` Запустим код. Ничего не произошло! Конечно, ведь мы всего лишь объявили функцию. Чтобы посмотреть результат, её необходимо вызвать. И делать это можно сколько угодно раз. `[4] hello()` `Hello world` `[5] hello()` `hello()` `hello()` `Hello world` `Hello world` `Hello world`	Для справки: Сайт: https://habr.com/ru/company/otus/blog/487952/ Перед уроком рекомендуется ознакомиться с материалами, представленными на сайте.

Функция, которую мы создали, довольно примитивная. Она заменяет собой однострочную функцию print(), что нерационально. Усложним задачу. Пусть она здоровается с тем, чьё имя мы зададим в программу.
Для этого понадобятся аргументы. Аргументы функции записываются в круглых скобках при её определении. Их может быть бесконечно много, но мы остановимся на одном:

```
[ ]  def hello(name):
       print("Hello "+ name)
```

Теперь, если мы просто вызовем функцию, произойдёт ошибка. Это логично, ведь функция ждёт входной аргумент — имя.

```
[2]  hello()
```

```
---------------------------------
TypeError
<ipython-input-2-a75d7781aaeb> in <
----> 1 hello()

TypeError: hello() missing 1 requir

SEARCH STACK OVERFLOW
```

```
[3]  hello("Петя")
```

Помимо принятия аргументов, функция может возвращать некоторое значение. Для этого используют ключевое слово return.
Давайте напишем функцию, которая ищет квадрат суммы.

```
[4]  def sum_squares(a,b):
       c = a**2 + 2*b*a + b**2
       return c
```

```
[5]  result = sum_squares(5,6)

     print(result)
```

```
121
```

В качестве аргументов здесь выступают сразу два целочисленных элемента. У функций, созданных вручную, нет ограничения на тип и количество принимаемых данных и возвращаемых значений. Попробуем отправить что-нибудь потяжелее. Например, массив.
Разработаем функцию, которая принимает на вход массив и вычисляет его среднее значение. Как мы отмечали ранее, работа с массивами идёт бок о бок с циклами, поэтому в теле функции мы будем использовать for.

```
[7]  def avg(x):
         summ = 0
         for i in x:
           summ=summ+i
         return summ/len(x)
```

Вопросы для обсуждения

Как вы думаете, может ли сложиться такая ситуация, когда у нас используются переменные с одним и тем же названием внутри и снаружи функции? Могут ли быть проблемы с этим? Можно ли обратиться из функции к переменной, которая находится «снаружи» или в другой функции?

Ответы школьников

До этого времени мы не говорили об области видимости переменных.
Понимание того, кто и когда может прочитать значение переменной, в Python упрощено. В сравнении с языком Си, это кажется небезопасным.

Переменные внутри функций имеют особый статус: они не видны остальной программе:

```
[3]  def building_piglet():
         piglet = "Наф-Наф"
```

```
print(piglet)
```

```
---------------------------------------------------------------
NameError                                 Traceback
<ipython-input-4-7b0d22ddba74> in <module>()
----> 1 print(piglet)

NameError: name 'piglet' is not defined
```

SEARCH STACK OVERFLOW

Действительно, переменная piglet незаметна для кода вне этой функции.

Скрытые переменные внутри функции называют локальными. А те, что находятся в основном коде программы, — глобальными. Глобальные переменные можно вызвать в любом участке программы — это их особенность.

Если где-то в программе мы создадим переменную второго поросёнка и попросим функцию её вывести, она это сделает.

Тем не менее, обращение функции к глобальной переменной — плохой тон в программировании. Передавать значение стоит по аргументу:

```
[11] piglet_2= "Ниф-ниф"
```

```
[12] def building_piglet():
         piglet = "Нар-Нар"
         print(piglet_2)
```

```
     building_piglet()
```

Почему есть жесткое разделение на глобальные и локальные переменные? Это сделали нарочно. В объёмных программах названия значений могут повторяться. Чтобы избежать ошибок, программисты отделили память, которую используют функции и память, в которой хранятся переменные. Это позволяет заводить переменные со схожими названиями:

```
[14] def print_number():

        number = 15
        print(number)
```

```
[15] number = -3

     print_number()
     print(number)
```

```
15
-3
```

Глобальные переменные доступны функциям только в режиме чтения. Переписать их мгновенно не получится. Чтобы «разморозить» режим записи, придётся воспользоваться ключевым словом global.

```
[19] piglet_2 = "Ниф-ниф"

     def building_piglet():
           piglet_2 = "Наф-Наф"

           print(piglet_2)
```

```
[20] building_piglet()

     print(piglet_2)
```

```
Наф-Наф
Ниф-ниф
```

Оно даст компилятору понять, что речь идёт именно о глобальной переменной:

```
piglet_2 = "Ниф-ниф"

def building_piglet():
      global piglet_2
      piglet_2 = "Наф-Наф"

      print(piglet_2)
```

```
[23] building_piglet()

     print(piglet_2)

     Наф-Наф
     Наф-Наф
```

В Python функции являются объектами, поэтому с ними можно работать, как с переменными.

```
def say_hello():
    print("Hello")


greetings = say_hello

greetings()
```

```
Hello
```

Например, положить результат работы функции в переменную или передать одну функцию в другую как аргумент.

В коде этой программы результат работы функции записан в переменную greetings. Позже мы обратимся к ней как к функции. Когда мы работаем с функцией как с переменной, ставить круглые скобки не нужно!

Мы увидели, что можем передавать функции как переменные. Используем это на практике.
При работе с данными часто используют библиотечную функцию map. В качестве одного аргумента она принимает массив значений. В качестве второго — функцию, которую необходимо применить к каждому элементу. Где это может быть полезно?

Для справки:
В Python анонимная функция — это функция, которая определяется без имени. В то время как обычные функции определяются с помощью ключевого слова def, анонимные определяются с помощью ключевого слова lambda

В задачах на вход программе может подаваться набор чисел через пробел: нужно посчитать их сумму, найти наименьшее и прочее. Как мы знаем, выходное значение функции input() — строка. Как сделать из неё массив чисел? Например, использовать циклы.

```
[2]  numbers = "123 142 79 66 300 55 32 56 32"
```

```
numbers = numbers.split()

for i in range(len(numbers)):
  numbers[i] = int(numbers[i])

print(numbers)
```

```
[123, 142, 79, 66, 300, 55, 32, 56, 32]
```

На помощь приходит функция map. В качестве первого аргумента выступает функция int, без скобок! Второй — знакомый нам массив.

```
numbers = map(int, numbers )

print(numbers)
```

Результат работы метода map() — неизвестный тип данных, который нужно снова превратить в список при помощи команды list().

```
[18] numbers = "123 142 79 66 300 55 32 56 32"
     numbers = list(map(int, numbers.split()))

     print(numbers)
```

Получившаяся запись лаконичнее кода, который мы записали в самом начале. Вы наверняка догадались, что вместо функции int() мы можем указать любую другую, даже созданную нами.

Рассмотрим другой пример. Пусть у нас есть список имён. Мы хотим получить вежливую форму обращения для каждого из них. С помощью той же функции map() мы можем применить функцию корректировки имён сразу ко всему списку одной строкой.

```
def polite_name(name):
    return 'Mr.' + name

guests = ["Boris","Ivan","Bob"]
guest_iterator = map(polite_name, guests)

list(guest_iterator)
```

```
['Mr.Boris', 'Mr.Ivan', 'Mr.Bob']
```

Все элементы списка изменились.

На практике функции для map() часто бывают короткими и используются только один раз. В этом случае удобно применять анонимные функции (или лямбда-функции). Процесс создания лямбда функции быстрее и лаконичнее, но их нужно использовать прямо в месте использования. Анонимные функции содержат лишь одно выражение, им необязательно давать имя. Они создаются с помощью инструкции lambda и имеют особый синтаксис.

Помните функцию для расчёта квадрата суммы? Давайте сделаем из неё лямбда-функцию. Для этого нужно указать имя, ключевое имя lambda, перечислить аргументы и записать через двоеточие возвращаемое значение.

```
[20] def sum_squares(a,b):
        c = a**2 + 2*b*a + b**2
        return c
```

```
[22] sum_squares = lambda a,b: a**2 + 2*b*a + b**2
```

В отличие от классических функций, анонимные всегда возвращают значение длиной в одну строчку и могут быть безымянными.

Наконец, давайте посмотрим, какой симбиоз они представляют совместно с map(). Запишем функцию polite_name проще и запустим программу.

		`` [27] guests=["Boris","Ivan","Bob"] guests = list(map(lambda name:"Mr."+name, guests)) [28] guests ``` Мы уместили объявление функции, её вызов и проход по массиву в одну строку! И всё благодаря map().	
Закрепление изученного материала	15 мин.	**Вопросы для обсуждения** • Для чего используют функции? • В чём отличие локальных переменных от глобальных? • Где можно использовать анонимные функции?	Педагог организует беседу по вопросам
Этап подведения итогов занятия (рефлексия)	8 мин.	**Вопросы для обсуждения** • Чему я научился? • С какими трудностями я столкнулся? • Какие вопросы у тебя остались? Что осталось непонятным?	Педагог способствует размышлению обучающихся над вопросами
Информация о домашнем задании, инструктаж по его применению	5 мин.	**Домашнее задание** представляет из себя решение нескольких задач по программированию. Программы проверяются в автоматическом режиме на платформе Stepik или на платформе Академии искусственного интеллекта.	

Рекомендуемые ресурсы для дополнительного изучения:

1. ПИТОНТЬЮТОР. [Электронный ресурс] – Режим доступа: http://pythontutor.ru/.
2. Онлайн игра на программирование CodeCombat:. [Электронный ресурс] – Режим доступа: https://codecombat.com/.
3. Прямая ссылка на начало игры. [Электронный ресурс] – Режим доступа: https://codecombat.com/play.
4. Области видимости в Python. [Электронный ресурс] – Режим доступа: https://habr.com/ru/company/otus/blog/487952/.