Кафедра: Цифровые технологии и прикладная информатика

ПРЕДМЕТ: «ОСНОВЫ ПРОГРАММИРОВАНИЯ 1»

ТЕМА 3. Функции 2

Преподаватель: доц. Аббасова Хатира

abbasova_xatira@unec.edu.az

Функции 2

ПЛАН ЛЕКЦИИ:

1. Определение функции
2. Комментарии Docstring
3. Примеры использования функций
4. Имена функций
5. Абстракция
6. Свойства
7. Фактические и формальные параметры функций
8. Псевдокод

На прошлом уроке

Категории ошибок в программах

- **Синтаксические ошибки -** известные как ошибки разбора кода. Синтаксический анализатор выводит строку, содержащую ошибку, и указывает место, где ошибка была обнаружена.
- **Семантические ошибки** - программа не работает так, как от нее ожидалось. Эти ошибки интерпретатор выявить не может, так как не владеет информацией о выполнении программы.

На прошлом уроке

- **Параметры** представляют собой локальные переменные, которым присваиваются значения в момент вызова (input) функции. А вводимые значения – это аргументы

```
>>> def summa(x, y):
        print(x + y)

>>> summa(4, 5)
9
>>>
```

параметры

аргументы

На прошлом уроке

- Определение функции определяет код который будет выполняться при вызове функции.
- Синтаксис для определения функции:

```
def имя_функции(параметр1, параметр2, ...):
    выражение
    выражение
    выражение
    ...
```

На прошлом уроке

- Определение функции состоит из двух частей:
- **Заголовок** (**Header**)
- **Тело** (**Body**)

Синтаксис: определение функции

- **Заголовок** (**Header**) – первая строка заголовка определения функции начинается со слова **def** (ingil. d. “define” - определить), затем следует имя функции и список параметров (если необходимо) в скобках.
- Скобки необходимы всегда, но не забывайте, что некоторые функции не нуждаются в параметрах.
- Заголовок заканчивается знаком двоеточия (**:**).
- Затем следует тело функции.

Синтаксис: определение функции

- **Тело** (**Body**) – состоит из выражений которые начинаются после отступа.
- Код начинающийся с отступом после двоеточия называется **блок**.
- Блок заканчивается строкой, после которой нет оступа.
- Внутри блока пустые строки допустимы.
- Тело функции это код который выполняется при вызове функции.
- В предыдущем примере строки **8-12** это тело функции **circle_at** ().

Код на Python - Круг

```
# circle_at.py
# функция для рисования окружности

from turtle import *

def circle_at(x, y, r):
    """Окружность с центром в тоске (x, y) и радиусом r"""
    penup()
    goto(x, y-r)
    pendown()
    setheading(0)
    circle(r)

circle_at(-200, 0, 20)
begin_fill()
circle_at(0, 0, 100)
end_fill()
circle_at(200, 0, 20)
hideturtle()
exitonclick()

```

Синтаксис: определение функции

- Определение функции обычно записывается в программах Python в верхней части после выражений инструкций импортирования, но до начала исполняющихся кодов.
- В предыдущем примере строки **14-20** выполняются как в программах которые мы писали ранее.

Комментарии Docstring

- Комментарии Docstring описывают функции другим программистам.
- Комментарии вводятся между тройными кавычками (три двойных кавычек) и могут содержать любое количество строк.
- Первая строка должна описывать цель функции, например в предыдущем примере **7-ая** строка описывает функцию **circle_at** ().
- Как и другие комментарии, Docstring комментарии не реализуются(не выпоняются).

Функция - примеры

- На Python для определения функции имеются ряд правил.
- Любые аргументы или входные параметры размещаются в круглых скобках **().**
- Первое выражение в функции по желанию может быть docstring.
- В Python функции определяются с помощью оператора def.
- За def следует имя функции. После имени функции ставятся скобки.
- Заголовок оканчивается двоеточием (:) и переходом на новую строку. Тело имеет отступ.
- После двоеточия следует тело, содержащее инструкции, которые выполняются при вызове функции.
- На следующем слайде записывая команду **func1()** мы определяем функцию **def func1 ()** и вызываем эту функцию. На экране печатается «**Я пишу код на Python**".

Функция - примеры

```
>>> def funk1():
        print("Я пишу код на Python")


>>> funk1()
Я пишу код на Python
>>>
```

Значение отступа в Python

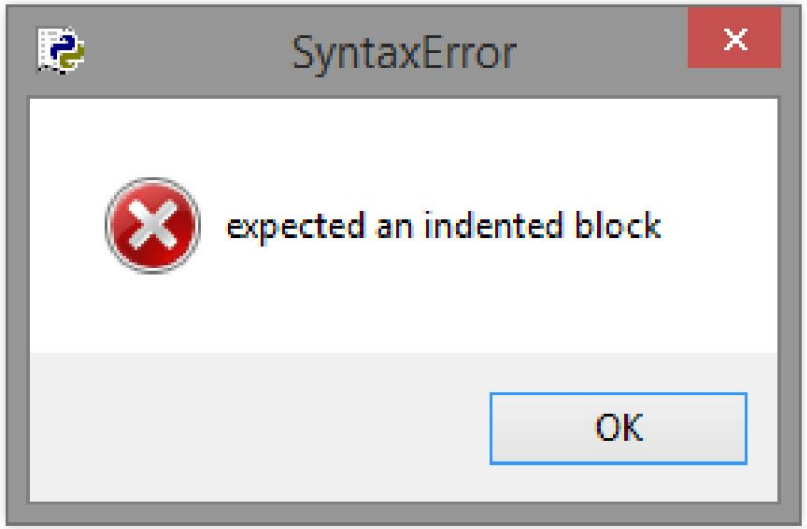
- Код в Python придерживается стилю с отступом.
- **В функциях Python не используют фигурные скобки для указания начала и конца блока, поэтому отступы в кодах имеют важное значение!**

Значение отступа в Python

Здесь мы используем простой пример с использованием функции "print " (печать).

При написании функции "**print**" под **def func 1()** всплывает сообщение с ошибкой: “**expected an indented block**".

```
def funk1():
print("Я пишу код на Python")
```



Значение отступа в Python

- А сейчас поставим отступ перед функцией "**print** " и на экран выводится сообщение: “Я пишу код на Python“ , как мы и ожидали.

```
>>> def funk1():
        print("Я пишу код на Python")


>>> funk1()
Я пишу код на Python
>>>
```

Значение отступа в Python

- Когда вы ставите отступы, вам также необходимо сохранить одинаковый стиль отступа для остальной части вашего кода,. Например, если мы напишем еще одну функцию **print**() в предыдущем примере не оставляя одинаковый отступ с первой функцией **print**(), тогда получим сообщение об ошибке **" "unindent does not match any other indentation level.”**

```
>>> def funk():
        print("Я пишу код на Python")
    print("код для funk1")

SyntaxError: unindent does not match any outer indentation level
>>>
>>>
```

Значение отступа в Python

Если оба выражения расположены с одинаковым отступом друг под другом, то функция возвращает нам ожидаемый результа

```
def funk1():
        print("Я пишу код на Puyhon")
        print("Код для funk1")

funk1()
```

```
Я пишу код на Puyhon
Код для funk1
>>>
```

Имена функций и параметров

- При создании собственной функции внимательно обдумывайте их имена и имена параметров.
- Постарайтесь выбрать имена, которые имеют смысл с точки зрения контекста, в котором они будут использоваться.
- Остерегайтесь очень коротких имен: имя **f (x)** почти всегда неопределенно, но в контексте **circle_at** () значения имен параметров **x**, **y** и **r** очень ясны.

Имена функций и параметров

- Единственное требование - имена функций и параметров должны быть допустимыми **идентификаторами** Python.
- Правила для идентификаторов находятся в Справочнике по языку Python, но для наших целей идентификаторы состоят из букв верхнего и нижнего регистра (включая неанглийские буквы), знака подчеркивания (_) и чисел вне первого символа (0-9).
- Существует небольшой набор отдельных **ключевых слов** (**keywords**) Python, которые нельзя использовать в качестве имен (например, слова from, import и def).

Абстракция: Функции помогают рассуждать

- Просмотрите строки **14-20** предыдущего примера. Когда вы используете функцию **circle_at (),** вам не нужно беспокоиться о том, как получить круг с центром в (x, y) - вы просто вызываете функцию.
- Это похоже на способ вызова библиотечных функций: мы не беспокоились о том, как работает **forward ()**, мы просто использовали эту функцию, чтобы переместить Перо turtle вперед.

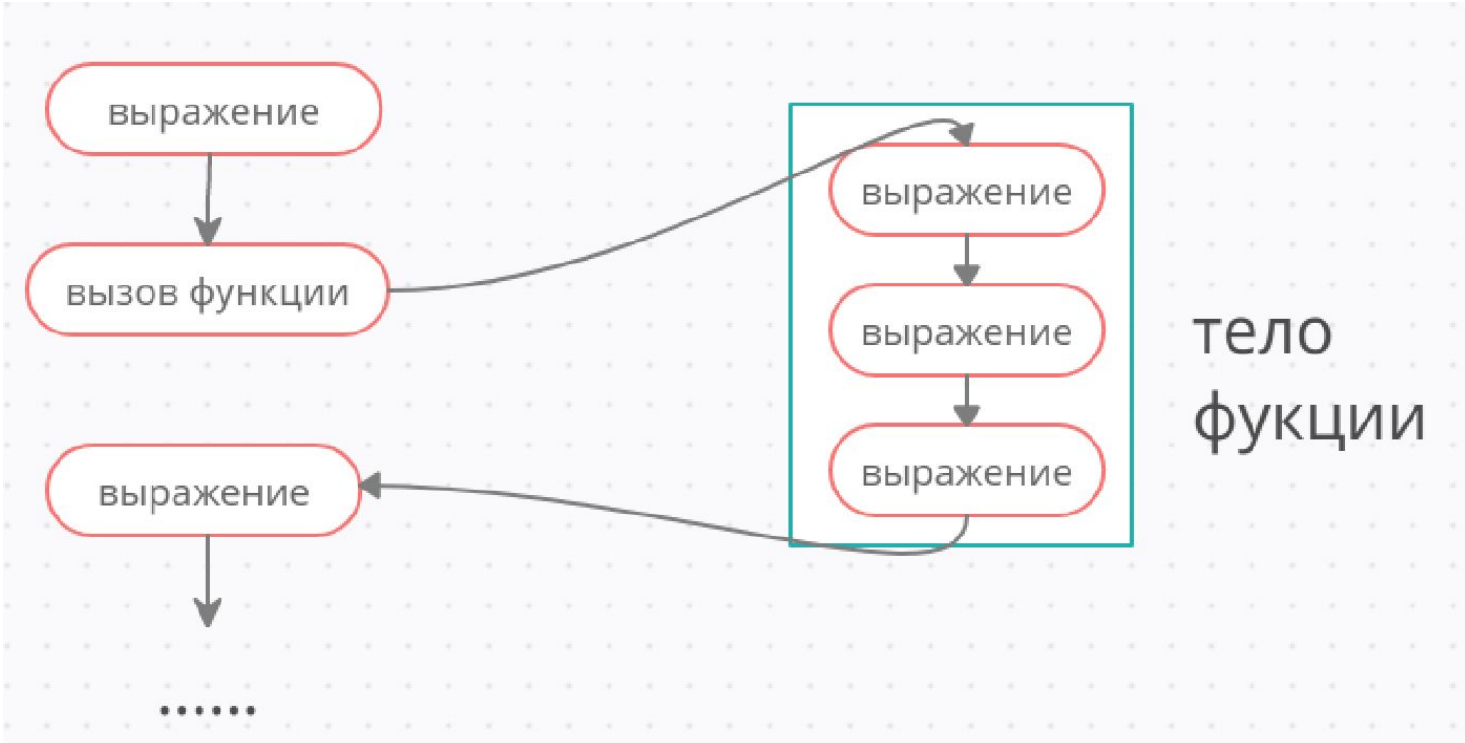
Абстракция : Функции помогают рассуждать

- Другими словами, написав новые функции, у нас есть возможность не видеть детали позже и думать на более высоком уровне, и в этом случае мы можем мыслить с точки зрения рисования окружности.
- Этот прием называется **абстракцией**. Абстракции позволяют нам не видеть деталей низкого уровня, чтобы мыслить в терминах концепций более высокого уровня.

Вызов функции изменяет поток управления

- Чтобы сделать абстракцию возможной изменяется поток управления программой.
- Как показано на следующем рисунке, для выполнения тела функции вызов функции создает отклонение в потоке управления.
- Это отклонение позволяет абстракции работать: вызов функции можно рассматривать как отдельный шаг (не задумываясь о деталях в теле функции).

Вызов функции изменяет поток управления



Свойства

- Рассмотрим несколько полезных особенностей при работе с функциями в Python.
- Имена функций в Python являются переменными, содержащими адрес объекта типа функция, поэтому этот адрес можно присвоить другой переменной и вызвать функцию с другим именем.

```
def summa(x, y):
    return x + y
f = summa
v = f(10, 3) #вызывается функция под другим именем
```

Свойства

- Параметры функции могут принимать значения по умолчанию:

```
def summa(x, y=2):
      return x + y
a = summa(3)  # вместо y подставляется значение по умолчанию
b = summa(10, 40)  # теперь значение второго параметра равно 40
```

Свойства

- Ранее мы сказали, что имя функции – обычная переменная, поэтому можем передать ее в качестве аргумента при вызове функции.
- Этот пример демонстрирует, как из функции **func** ()можно вызвать функцию **summa**().

```
def summa(x, y):
    return x + y
def func(f, a, b):
    return f(a, b)
v = func(summa, 10, 3) # передаем summa в качестве аргумента
```

Свойства

- В момент вызова функции можно присваивать значения конкретным параметрам (использовать ключевые аргументы).
- Ошибкой будет являться вызов функции, при котором не задан аргумент **a**, т.к. для него не указано значение по умолчанию.

```
def func(a, b=5, c=10):
    print('a равно', a, ', b равно', b, ', a c равно', c)
func(3, 7)          # a=3, b=7, c=10
func(25, c=24)      # a=25, b=5, c=24
func(c=50, a=100)   # a=100, b=5, c=50
```

Внутренние функции

- Python позволяет определять функцию внутри другой функции:

```
>>> def outer(a, b):
      def inner(c, d):
            return c + d
      return inner(a, b)

>>> outer(4, 7)
11
>>>
```

Функции

- Функции «инкапсулируют» задачу (объединяют множество инструкций в одну строку кода). Большинство языков программирования имеют множество встроенных функций (в противном случае для выполнения одних и тех же задач потребовалось бы много шагов), таких как вычисление квадратного корня из числа. В общем, пользователю не интересно знать, как работает функция, ему нужно только справиться с необходимой работой!
- Когда функция «вызывается», программа «покидает» текущую часть кода и начинает выполнять первую строку внутри функции. Таким образом, «поток управления» функции:
- 1 - Программа переходит к строке кода, содержащей «вызов функции».
- 2 - Программа входит в функцию (начиная с первой строки кода функции).
- 3 - Все инструкции внутри функции выполняются сверху вниз.
- 4 - Программа выходит из функции и возвращается туда, где она была запущена.
- 5 - Любая информация, вычисленная и ВОЗВРАЩЕННАЯ функцией, используется вместо функции в исходной строке кода.

Зачем писать функции?

- 1 - Они позволяют нам описать нашу программу в виде нескольких подэтапов. (Каждый подэтап может быть самостоятельной функцией. Если какая-либо программа кажется слишком сложной, разбейте всю программу на подэтапы!)
- 2 - Они позволяют нам повторно использовать код вместо того, чтобы писать его с нуля.
- 3 - Функции позволяют нам содержать пространство имен переменных в чистоте (локальные переменные «работают» только во время работы функции). Другими словами, function_1 может использовать переменную с именем i, а function_2 может также использовать переменную с именем i, и здесь нет путаницы. Каждая переменная существует только тогда, когда компьютер выполняет заданную функцию.
- 4 - Возможности позволяют тестировать небольшие части нашей программы изолированно от остальных.

Шаги по написанию функции

1 - Понять назначение функции.

2 - Назначьте данные, введенные в функцию вызывающей функцией (в виде параметров)!

3 - Определите, какие переменные данных необходимы в функции для достижения цели.

4 - Определитесь, какие шаги программа будет использовать для выполнения конкретной задачи. (Алгоритм)

Части «черного ящика» (т.е. функции)

- Функции можно назвать «черными ящиками», потому что вам не нужно знать, как они работают - только то, что вводить и что выводить.
- При определении программы как черного ящика мы должны описать следующие атрибуты функции.
- Примечание. Большинство систем документации, отражающих возможности функций, имеют только следующий формат - только атрибуты функции, связанный код не отображается.
- 1 - Имя - описывает назначение функции. Например, глагол или предложение, такое как «средний_вычислить» или просто «средний».
- 2 - Входы - называются параметрами: укажите, какая информация необходима для работы функции, и дайте символическое имя каждой части информации, которая будет использоваться в функции.
- 3 - Расчет - различается для каждой функции
- 4 - Выход - значение (но иногда нулевое, а иногда и несколько), которое обычно вычисляется внутри функции и «возвращается» с помощью выходных переменных.

Рабочее пространство функции

- Каждая функция имеет собственное рабочее пространство. Это означает, что каждая переменная в функции может использоваться только во время выполнения функции (а затем переменные исчезают).
- Для правильной разработки программного обеспечения важно иметь отдельное «рабочее пространство» для каждой функции. Если бы каждая функция совместно использовала каждую переменную во всей программе, было бы легче изменить значения некоторых переменных, не изменяя их. Кроме того, было бы трудно вспомнить, какие «имена» использовались где-либо еще, и было бы сложно найти новые имена, отражающие аналогичные идеи.
- Побочный эффект несуществующих переменных функции после истечения срока действия функции заключается в том, что единственный способ «выйти за пределы» функции - «вернуть» эту информацию в вывод функции.
- Кроме того, функция может только «видеть» данные, «переданные» ей через параметры. Таким образом, единственный способ «ввести» информацию в функцию - это использовать параметры.

Формальные и фактические параметры

- Когда мы создаем функцию, она должна представлять «общее» действие, которое можно применить ко многим ситуациям. Например, если мы хотим узнать среднюю оценку, неважно, какому тесту, викторине, заданию и т. д. ... он принадлежит? Если нам задали какой-нибудь прайс-лист, который можем оценить, мы сможем вычислить среднее значение!
- ... но если может быть какой-то прайс-лист, как узнать название прайс-листа? Ответ: Неважно.
- Вы, как программист функции, даете данным имя по своему выбору. Это похоже на то, как продавец звонит вам и пытается продать что-то по сценарию: Уважаемый _ введите сюда свое имя клиента_, позвольте мне продать вам наш прекрасный продукт.

Псевдокод

- Вот пример функции псевдокода:

функция average_price (список_цен)

....

конец функции

Псевдокод

функция average_price (список цен)

....

конец функции

В функции average_price имя price_list будет использоваться вместо любой переменной, содержащей прайс-лист, созданный другим пользователем. Чтобы вызвать функцию, вы можете написать что-то вроде этого:

// другой код (не связанный с нашей функцией)

semestr_price = ... // создаем массив цен

print "Средняя оценка за семестр:"

распечатать average_price (semester_price)

Псевдокод

- В этом коде цены хранятся в переменной «semester_price». Внутри функции цены хранятся в переменной «price_list». Таким образом, во время выполнения программы, оба имени будут ссылаться к одному и тому же объекту, но в разное время.
- Параметр «price_list» называется формальным параметром; опять же, это заполнитель для любого набора цен.
- Переменная semester_value называется фактическим параметром. То есть он указывает, «что вы фактически используете» для вызова функции.

Литература:

1. Gowrishankar S, Veena A, Introduction to Python Programming, CRC Press, 2019, Taylor & Francis Group
2. Билл Любанович, Простой Python. Современный стиль программирования, Питер 2019
3. Д. Златопольский Основы программирования на языке Python. – М.: ДМК Пресс,
4. 2017. – 284 с
5. С. К. Буйначев, Н. Ю. Боклаг, Основы программирования на языке python, Екатеринбург Издательство Уральского университета 2014
6. Д. Ю. Федоров, Основы программирования на примере языка Python, Учебное пособие, Санкт-Петербург 2019
7. С. Шапошникова, Основы программирования на Python, http://younglinux.info, 2014
8. Rəşad Qarayev, Python programlama dili, Bakı – 2015
9. Abdulla Qəhrəmanov, İlahə Cəfərova, Python proqramlaşdırma dili, Bakı – 2015

Спасибо за внимание!