**PyCharm:**

Alt + CMD + L – форматирование кода PEP 8

Shift + Alt + выделение строк – одновременное редактирование выделенных строк.

Shift + CMD + -\+ - свернуть\развернуть код

**Идентификаторы:**

это имена, используемые для обозначения переменной, функции, модуля, класса или другого объекта.

* Классы – с большой буквы *«Eggs»*
* Закрытый или вне класса – с нижним подчеркиванием «\_eggs»
* Спец имя определенное внутри языка – нижнее подчеркивание в начале и в конце «\_eggs\_»

Идентификаторы не могут повторять ключевые слова.

В данной таблице собраны все **ключевые слова Python**.

|  |  |  |  |
| --- | --- | --- | --- |
| False | class | from | or |
| None | continue | global | pass |
| True | def | if | raise |
| and | del | import | return |
| as | elif | in | try |
| assert | else | is | while |
| async | except | lambda | with |
| await | finally | nonlocal | yield |
| break | for | not |  |

**If** - если «это\_1»

**Elif** - либо «это\_2»

**Else** - если не if и elif, то «это\_3»

**Операторы:**

*2 + 3**«2» и «3» - операнды*

*«+» - оператор*

* Арифметические операторы
* Операторы сравнения (реляционные)
* Операторы присваивания
* Побитовые операторы
* Логические операторы
* Операторы членства (Membership operators)
* Операторы тождественности (Identity operators)

**Арифметические операторы**

|  |  |  |  |
| --- | --- | --- | --- |
| **+** | сложение | суммирует значение правого и левого операнда | 15 + 5 = 20 |
| **-** | вычитание | вычитает правый операнд из левого | 15 – 5 = 10 |
| **\*** | умножение | перемножает операнды | 5 \* 5 = 25 |
| **/** | деление | делит левый операнд на правый | 25 / 5 = 5 |
| **%** | деление по модулю | делит левый операнд на правый и возвращает остаток | 7 / 2 = 1  (= 3 и остаток 1) |
| **\*\*** | возведение в степень | возведение левого операнда в степень правого | 5 \*\* 2 = 25 (5 \* 5) |
| **//** | целочисленное деление | возвращается целая часть результата, результат после запятой отбрасывается | 7 // 2 = 3 |

**Операторы сравнения**

|  |  |  |  |
| --- | --- | --- | --- |
| **==** | равенство  да = True | Проверяет равны ли оба операнда. Если да – результат «True» | 5 == 5 #True  3 == 5 #False |
| **!=** | равенство  да = False | Проверяет равны ли оба операнда. Если да – результат «False» | 5 != 5 #False  3 != 5 #True |
| **<>** | равенство  нет = True | Проверяет равны ли оба операнда. Если нет – результата «False» (python 2.7) | 5 != 5 #False  3 != 5 #True |
| **>** | Л больше П | Проверяет больше ли значение левого операнда, чем значение правого операнда. Если да – результат «True» | 5 > 3 #True  True > False # True |
| **<** | Л меньше П | Проверяет меньше ли значение левого операнда, чем значение правого. Если да – результат «True» | 3 < 5 #True  True < False # False |
| **>=** | Л больше или равен П | Проверяет больше или равно значение левого операнда, чем значение правого. Если да – результат «True» | 5 >= 3 #True  -3 >= -2 #False |
| **<=** | Л меньше или равен П | Проверяет меньше или равно значение левого операнда, чем правого. Если да – результат «True» | 4 <= 5 #True  0 <= 0.0 #True |

**Оператор присваивания**

C = 5; A = 2;

|  |  |  |
| --- | --- | --- |
| **=** | Присвоение значения правого операнда левому | С = 5 #5 |
| **+=** | Прибавление значения правого операнда к левому | C += A #7 |
| **-=** | Вычитание значения правого операнда из левого | C -= A #3 |
| **\*=** | Умножение значения левого операнда на правый | С \*= A #10 |
| **/=** | Деление значения левого операнда правый | С(10) /= А(5) #2 |
| **%=** | Деление операндов по модулю | C %= A #1 |
| **\*\*=** | Возведение левого операнда в степень правого | С \*\*= А #25 |
| **//=** | Целочисленное деление левого операнда на правый | С (11) //= А #5 |

**Побитовые операторы**

**Типы данных:**

bool (true/false) – ДА или НЕТ

Int (4) - целое число (INTEGER)

Float (3.14) - дробное число (точность 15 знаков после .)

Complex (1jˆ2 = -1) - комплексное (не знаю)

Str («Любой текст») - строка (STRING)

**Циклы:**

**While** - пока «eggs» не станет ниже 0 – будет выводиться параметр «eggs», а затем отниматься -1 и повторяться цикл.

*eggs = 3*

*while eggs <= 0:*

*print (eggs)*

*eggs = eggs -1*

Бесконечный цикл «while 1 == 1», пока число не дойдет до 10.

*eggs = 1*

*while 1 == 1:*

*print (str(eggs) + « eggs»)*

*eggs += 1*

*if eggs = 10*

*break*

**Break** – принудительная остановка

**continue** - прододжить

**Техники:**

Комментирование «#» - текст до конца строки не будет интерпретирован

*#бла бла бла любой текст*

*print («А вот этот текст уже будет отображен! »)*

Экранирование «\» - отключает исполнение идущего следом синтаксиса.

*Print: («Hello \«World!\»») = Hello «World!»)*

Перенос строки «\n» - переносит следом идущую информацию на др. строку.

*Print: («Hello \nWorld!»)*

Многострочные текста «’’’»

*Print: ’’’текст \*пробел\* текст’’’*

Преобразование типа данных

Для добавления одного типа данных к другому нужно привести их к единому типу данных.

*Int: Number = 27  
Str: print = («Мне str(number) лет)*

Конкатенация

Добавление одной строки к другой

*Print («Ахаха» + «охохо»)*

Умножение строк

Повторение необходимого элемента нужное кл-во раз

*Print («Привет! » \* 5) – Повторит «Привет!» 5 раз*

*Print (5 \* «Привет! »)– Повторит «Привет!» 5 раз*

**Функции:**

Print – вывести текст

*print («текст»)*

Input – ввести текст

*input («Как Вас зовут? »)*

Переменные – блок с данными, которыми я могу оперировать

*Имя переменной = значение переменной age = 27  
 (имя не может начинаться с цифры, и не должна содержать спецсимволы, чувствительный к регистру!)*

Удаление переменной – ниже этой команды переменная не будет работать.

*Del имя\_переменной*

Метасинтаксические переменные *– «foo» «bar» «eggs» «spam» - имена переменных по умолчанию в туториалах.*

*Методы списков:   
upper(), .title() и .lower() корректируют регистр строки.*

*.split() берет строку и создает список подстрок.*

*.join() берет список строк и создает строку.*

*.strip() очищает начало и конец строки от пробелов и других помех.*

*.replace() заменяет все символы/строки в строке на другие символы/строки.*

*.find() ищет в строке символ/строку и возвращает значение индекса, по которому найден символ/строка.*

*.format() позволяет интерполировать строку с переменными.*

*Итерирование 2 списков в словарь*

drinks\_to\_caffeine = {drinks:caffeine for drinks, caffeine in zip(drinks, caffeine)}

*Присвоение нового ключа с параметром в словарь spread*

spread["present"] = 5

*Присвоение ключа 22 из словаря tarot в словарь spread, ключ present. С последующим удалением ключа 22 из словаря tarot*

spread["present"] = tarot.pop(22)

Функция разбивает словарь на 2 части (ключ а и значение b, а дальше подставляет в строку с помощью метода format.

for a, b in dict.items():

print(“бла {} бла {} бла”.format(a, b))

Открытие документа:

with open("name.txt", “a”(r/w/a)) as name\_variable:

Запись листа со словарями в CSV файл (таблицу)

# наш лист со словарем

access\_log = [{'time': '08:39:37', 'limit': 844404, 'address': '1.227.124.181'}]  
# лист с заголовками

fields = ['time', 'address', 'limit']

# импортируем модуль для работы с csv файлами

Import csv  
# создаем csv файл

with open("logger.csv","w") as logger\_csv:  
# создаем функицю cvs.DictWriter(куда записывать, по каким ключам)

log\_writer = csv.DictWriter(logger\_csv, fieldnames=fields)

# записываем заголовки из fields в logger\_csv

log\_writer.writeheader()

# итерируем каждый словарь из списка access\_log и записываем его в logger\_csv

for i in access\_log:

log\_writer.writerow(i)

Запись словаря Python в JSON файл:

data\_payload = {'interesting message': 'What is JSON? A web application\'s little pile of secrets.',

'follow up': 'But enough talk!'}

import json  
# создаем json файл (data.json) в режиме записи ("w")

with open("data.json", "w") as data\_json:

# с помощью метода dump из библиотеки json добавляю словарь в файл data.json

json.dump(data\_payload, data\_json)

Методы для файлов:

Открывать объекты файлов с помощью open() и with.

Читать полное содержимое файла с помощью метода Python .read().

Читать файл построчно, используя .readline() и .readlines().

Создавать новые файлы, открывая их в режиме записи.

Неразрушающее добавление в файл путем открытия файла в режиме append-mode. Open(name.csv, “a”)

**Назначение и использование класса:**

# создаем класс Dog

class Dog:  
# присваиваем переменные и функции внутри класса. В функции обязателен 1 атрибут (self)  
  dog\_time\_dilation = 7  
  
  def time\_explanation(self):  
    print("Dogs experience {} years for every 1 human year.".format(self.dog\_time\_dilation))  
  
# присваеваем класс Dog переменной pipi\_pitbull  
pipi\_pitbull = Dog()

# вызываем переменную pipi\_pitbull.имя\_функции\_в\_классе (теперь это метод)  
pipi\_pitbull.time\_explanation()  
# Prints "Dogs experience 7 years for every 1 human year."

isinstance(объект, класс) – проверка объекта на принадлежность к классу. Вернет True или False