"""

Типы данных делятся на:

Изменяемые - list, dict, set

Неизменяемые - int, float, str, frozenset, tuple, bool, None

Numbers

числа - неизменяемый тип данных, предназначенный для хранения значении и проведении каких-либо операции над ними

int -> (целочисленные)

float -> (числа, с плавающей точкой)

complex -> (комплексные числа)

decimal -> (десятичные числа)

long -> (огромные числа)

abs(int)

abs(float)

divmod(num1, num2) -> (num1//num2),(num1%num2)

возвращает нам 2 результата в виде tuple

pow(num, degree) - num \*\* degree

pow(num, degree, ostatok) - num \*\* degree -> % ostatok

round(num, digits) -> округляет число, если мы передадим второе число,

то округлит

еще столько символов после точки

round(5.7) -> 6 (округляет в большую сторону)

round(12.056, 2) -> 12.06

Strings(Строки) - неизменяемый тип данных, который представляет собой

упорядоченную последовательность, заключенных в двойные либо одинарные

кавычки

Синтаксис:

str = 'python'

str = "python"

str1 = "hello"\*3 -> hellohellohello

str2 = "hello" + "world" - > "helloworld" #конкатенация

индексы - нумерация символов в упорядоченной последовательности

(индексация начинается с 0)

срезы - часть последовательности(подпоследовательность)

string = "hello world"

string[:-1] -> "hello worl"

string[:5] -> "hello"

Bool - логический тип данных, с двумя значениями, ипользуется в условиях

TRUE - правда

FALSE - ложь

bool( [3, 2, 1]) -> True

bool([]) -> False

bool([[]]) -> True

List - изменяемый и упорядоченный тип данных, который хранит в

себе последовательность элементов. Элементами списка могут быть любые

типы данных, в том числе и списки

list1 = [1, "str", 3.4, (56,7), ["hello", True]]

list2 = list((1, 2, 3, 4))

len(sequence) - возвращает длину последовательности (количество элементом)

len(["hello"]) -> 1

range(start, end, step) - генерирует последовательность чисел

list1 = list(range(10)) -> 0-9

Кортеж - неизменяемый тип данных

(1, 2, 3, 4) == 1, 2, 3, 4

a = (3) - int

b = (3,) - tuple

c = 3, - tuple

d = (1,2,3) - tuple

Set - изменяемый, неупорядоченный тип данных, который хранит в себе только

уникальные неизменяемые значения

Синтаксис:

{1, 2, 3, 4, 5}

set(iterable)

Dict - словарь, изменяемый, неупорядоченный тип данных, в котором хранятся

пары в виде ключ - значение. Ключ в словаре - любой неизменяемый и уникальный значения отделяется двоеточием.

Синтаксис:

{"key": "value"}

dict(iterable)

Loops

цикл - блок кода, который будет выполнятся несколько раз. Каждый круг цикла называется итерацией.

break - инструкция, которая немедлено прерывает работу цикла

continue - инструкция, которая немедлено переходит к следующей итерации

For

for - цикл, который производит итерации над последовательностью (list, dict, set, str, tuple) в цикле for мы можем выполнять различные операции над каждым элементом последовательности

Синтаксиc: for элемент in последовательность: какие-то действия

While

while - цикл, который производит какие-то действия пока условие будет True

Синтаксиc: while условие: какие-то действия

Conditions

условия - оператор, который позволяет нам выполнять или не выполнять какой-то кусочек кода, который находится в теле условия.

Синтаксиc: if условие: тело

if True: print("Условие верное")

Также мы можем создавать конструкции из условных операторов.

if условие: тело1 elif условие: тело2 else: тело3

если первое if условие выдало True, то выполнится 'тело1'

если первое if условие выдало False, то проверяется второе условие elif

если второе elif условие выдало True, то выполнится 'тело2'

если второе elif условие выдало False, то выполняется 'тело3'

if и else в конструкции может быть использовано только 1 раз, elif может быть много

Тернарные условия

тернарное условие - условие написанные в одну строку, которое возвращает одно из двух результатов в зависимости от условия

тело1 if условие else тело2

a = "Hello" if False else "Bye"

a == "Bye"

Comprehensions

генератор последовательности

Синтаксис: действие for элемент in последовательность [if условие]

базовые comprehensions

list\_ = [1,2,3,4,5]

a = (x for x in list\_) -> генератор

list\_copy = list(a) -> список

a = ((x, x\*\*2) for x in list\_) -> генератор

dict\_ = dict(a) -> словарь

по новому формату

[x for x in list\_] -> список

{x : x\*\*2 for x in list\_} -> словарь

comprehensions с фильтрами

[x for x in list\_ if not x%2]

# res = []

# for x in list\_:

# if not x%2:

# res.append(x)

comprehensions с тернарными условиями

["нечетное" if x%2 else "четное" for x in list\_]

# res = []

# for x in list\_:

# if x%2:

# res.append("нечетное")

# else:

# res.append("четное")

Try-except

try-except - конструкция, для обработки исключений и ошибок. Синтаксические ошибки мы не можем обработать

Синтаксиc:

try:

код, который может вызвать ошибку

except Исключение:

код, который работает, если ошибка вышла

else:

код, который работает, если ошибка не вышла

finally:

код, который работает в любом случае (даже когда код сломался)

Functions

функция - именованный блок кода, выполняющий какие-то действия и возвращающий какой-то результат. Мы можем вызывать функцию, обращаясь к ней по имени и используя круглые скобочки. код, который написан внутри функции будет работать только при вызове функции функции могут принимать данные

"параметры функции" - локальные переменные, которым присваются значения при вызове функции

"аргументы функции" - конкретные значения, которые мы передаем в параметры функции

"def" - инструкция, с помощью которой определяется функция

"return" - инструкция, с помощью которой функция возвращает результат, если ее не прописать (по дефолту возвращается значение None)

Функция высшего порядка — это функция, принимающая в качестве аргументов другие функции или возвращающая другую функцию в качестве результата.

Types of arguments

позиционные аргументы

именованные аргументы

параметры с дефолтом

необязательные аргументы (\*)

ключевые аргументы (\*\*)

def func(позиционные, с\_дефолтом, \*args, \*\*kwargs):

args - tuple

kwargs - dict

func(позиционные, необязательные\_аргументы, ключ=значение)

Lambda

lambda - анонимная функция

Синтаксис:

# lambda параметры: возвращаемый результат или действие

myfunc = lambda x, y: (x\*\*2, y\*\*2)

print(myfunc(5, 6))

Встроенные функции

map

map - функция, которая выполняет действия над каждым элементом последовательности и возвращает генератор новой последовательности она принимает 2 аргумента:

функция, которая принимает 1 аргумент

последовательность

Синтаксиc:

res = map(int, ["1", "2", "3"])

# int("1") -> 1

# int("2") -> 2

# int("3") -> 3

res = list(res)

res == [1, 2, 3]

def myfunc(elem):

return elem.upper()

res = map(myfunc, ["hello", "world"])

# myfunc("hello") -> "HELLO"

# myfunc("world") -> "WORLD"

res = list(res)

res == ["HELLO", "WORLD"]

filter

filter - функция, которая возвращает последовательность из элементов, соответствующих условию она принимает 2 аргумента:

функция, которая принимает 1 аргумент, которая возвращает булевое значение (если True, то добавляется)

последовательность

Синтаксиc:

def myfunc(elem):

return elem.isalpha()

res = list( filter(myfunc, ["1", "d", 't']) )

# myfunc("1") -> False

# myfunc("d") -> True

# myfunc("t") -> True

res == ["d", "t"]

def myfunc(elem):

return elem > 0

res = list( filter(myfunc, [-1, 4, 0, 8, 30, -2]) )

# myfunc(-1) -> False

# myfunc(4) -> True

# myfunc(0) -> False

# myfunc(8) -> True

res == [4, 8, 30]

res = list( filter(lambda elem: elem > 0, [-1, 4, 0, 8, 30, -2]) )

res == [4, 8, 30]

res = list( map(myfunc, [-1, 4, -6, 0, 8, 30, -2]) )

# myfunc(-1) -> False

# myfunc(4) -> True

res == [False, True, False, False, True, True, False]

Zip

zip - функция, которая обьединяет элементы из нескольких последовательностей по индексам в tuple, т.е все элементы под индексом 0 в первый tuple, все элементы под индексом 1 во второй tuple и т.д

list1 = [[1], [2], [3], [4], [5]] # 5

list2 = ["a", "b", "c", "d"] # 4

list3 = [1.0, 2.0, 3.0] # 3

res = list( zip(list1, list2, list3) )

res == [([1], 'a', 1.0), ([2], 'b', 2.0), ([3], 'c', 3.0)]

Reduce

reduce - функция, которую надо импортировать из библиотеки functools from functools import reduce она принимает 2 аргумента:

функция, которая принимает 2 аргумента

последовательность

функция reduce берет из последовательности 2 элемента, отправляет их в функцию. результат и следующий элемент из последовательности снова отправляет в функцию и т.д

Синтаксиc:

from functools import reduce

def myfunc(x, y):

return x if len(x) >= len(y) else y

res= reduce(myfunc, ["hello", "world", "makers"])

res == "makers"

res = reduce(lambda x,y: x if len(x) >= len(y) else y, ["hello", "world", "makers"])

res == "makers"

Области видимости делятся на:

# BUILT\_IN

# GLOBAL

# ENCLOSED

# LOCAL

#globals() - возвращает ссылку на текущий словарь глобального пространства имен. Ее можно использовать для обращения к объектам в этом пространстве

#locals() - возвращает ссылку на словарь локального пространства имен. Обращается к объектам локального пространства имен

global - для того чтобы перезаписать то, что находится на глобальном уровне

nonlocal - для того чтобы перезаписать то, что находится на замкнутом пространстве

"""

# a = {'1': 23, '2': 54}

# print(a.get('1', default='There is no such key')) # 23

# print(a['3']) # KeyError

# TypeError

# 2 + 'hello'

# ValueError

# int('hello')

# IndentationError, SyntaxError

# def hello():

# asdfasdf

# try:

# some code

# except:

# another action

# else:

# another code

# finally:

# works always

# def hello():

# for i in range(1, 10):

# print(i)

# DRY - dont repeat yourself

# def hello():

# return 10

# num = hello() # 10

# print()

# input()

# set()

# len()

# type()

# str()

# filter(func, iterable) -> filtered object

# lambda

# map(func, iterable) -> muted object

# min(), max(), sum()

# reduce(func, iterable) #from functools import reduce

# lambda x, z, y: x + z + y

# def add(x, z, y):

# return x + z + y

# for element in iterable\_object

# while condition

# break

# continue

# [i for i in range(10)]

# [action loop]

# [i for i in range(10) if not i % 2]

# [action loop condition]

# BUILT\_IN

# GLOBAL

# ENCLOSED

# LOCAL

# a = 20

# print(locals())

# def hello():

# a = 10

# print(globals())

# hello()

# def hello(\*a, \*\*b): # params

# print(a)

# print(b)

# return a, b

# print(hello(10, 20, 40, 20, i=70)) # arguments