Что сделать: справочник в другом формате. Может chm? Может html?

Видео писать не как видео, а как слайдшоу с голосом. Там видео не нужно. Чтобы можно было далее нажимать.

И, главное, видно было текст.   
Как сохранять с сохранением качества?

Составить список уроков. Темы, какие вопросы на каждом уроке.

1. Установка
2. Настройка
3. Первая программа
4. Решение всех задач
5. Приемы работы с клавиатурой

Может сразу на progras.ru?

Термины

**Операторы**:  
 математические операторы: + - \* / % = и т.д.  
 логические операторы: not, or, and, xor  
 операторы отношения: <, !=, ==, >, <=, >=, in  
 оператор for, while, if, …

**Переменная**: область в памяти, которая имеет имя.

**Последовательность**: список, кортеж, словарь. Условно: одна переменная имеет несколько значений. Обращение к переменной либо по номеру ячейки (список, кортеж), либо по названию ячейки (словарь).

**Команда** – тоже, что и оператор

**Функция** – участок кода, имеющий свое название и свою область переменных. Может возвращать значение (как sin(x)), а может и производить действия, не возвращая значения (как print()). Даже не получая аргументов нужно вызывать со скобками.

**Метод** – функция, которая «принадлежит» объекту. Всегда вызывается через указание имени метода через точку, после имени объекта, например: a.sort() – метод списка для сортировки списка.

**Параметр, аргумент** – значение или переменная, которая передается в функцию. Аргумент – это значение, которое передано функции. Параметр – принятый функцией агрумент.

**Исключение**  - «повод» программе прекратить работу и «упасть с ошибкой», т.е. информацией о возникшей исключительной ситуации, препятствующей дальнейшей работе. Деление на ноль, попытка открыть несуществующий файл, попытка взять значение из несуществующей переменной, выход за пределы списка и т.д. Исключения можно перехватывать и обрабатывать, тогда они не приводят к прекращению работы программы.

Английские слова

width - ширина

height – высота

length – длина

for – для

while - пока

if – если

else – иначе

define - определить

error - ошибка

indented block - отступ

expected - требуется

function – функция (подпрограмма)

text - текст

line – линия (графический объект)

rectangle – прямоугольник (графический объект)

triangle - треугольник

circle – окружность (графический объект)

oval – овал (графический объект)

create - создать

delete - удалить

find - найти

index – порядковый номер (индекс)

break – прервать

root – главный

after – после

time – время

integer –целые числа

float – вещественные числа (числа с плавающей запятой)

string – строка

pass – пропустить (ничего не далать)

event – событие

button- кнопка

key – клавиша на клавиатуре

mouse – мышь

keyboard - клавиатура

click – нажать и отпустить кнопку мыши (сделать щелчок, щелкнуть)

release – отпустить кнопку мыши

color - цвет

loop - цикл

bind- связать

new - новый

old - старый

ball – мяч

all – все

in – в (предлог)

not - не (отрицание)

fill – заливка

canvas – холст (полотно) для рисования

Приемы работы с клавиатурой и редактором

Ctrl+<клавиши управления курсором> - переход по словам влево и вправо

End, Home – перейти в начало, в конец строки

Tab, Shit+Tab – увеличить, уменьшить отступ (в IDLE – Ctrl+[, Ctrl+])

Ctrl+N – создать новый файл (вкладку в Geany)

Ctrl+F - искать текст в файле

Ctrl+C, Ctrl+Ins – скопировать текст

Ctrl+V,Shift+Ins вставить скопированный текст

F5 – запуск программы

Прием: быстрое копирование строк текста (тут должно быть видео)

Как и где искать информацию

ссылки на правильные сайты:  
python.org

<http://www.ibm.com/developerworks/ru/library/l-python_part_1/index.html>

<http://younglinux.info/python.php>   
Tkinter: <http://effbot.org/tkinterbook/> (En)(самый полный сайт)

<http://pythonworld.ru/>

<http://habrahabr.ru/post/205944/>

<http://habrahabr.ru/post/61905/>

приемы поисковых запросов:

Например, нужно узнать, как округлить число: python округлить до знаков

Или разобраться с установкой Geany: python geany не запускается

получить список доступных функций и констант:

print(dir())

print(dir(\_\_builtins\_\_))

узнать, какие функции и константы доступны в модуле random:

**import random**

**print(dir(random))**

узнать возможности функции или метода, посмотреть на краткий пример использования:

print(set.\_\_doc\_\_)

print(''.join.\_\_doc\_\_)

from turtle import \*

print(fd.\_\_doc\_\_)

Установка Python и первая программа

отдельный файл

Математические операторы и логические выражения

a = 10 - изменить значение

a -= 10 - уменьшить значение на 10

a += 10 – увеличить значение на 10

c = a + 5 - в переменную с поместить значение на 5 больше, чем в переменной a

c = a \* 2 – в перменную с поместить значение, в 5 раз большее, чем в переменной а

c= a / 3 – разделить а на 3 и результат поместить в переменную с. Результат: float

c = a // 3 - разделить а на 3, отбросить дробную часть, результат (целое число) поместить в с. Результат: int

c = a % 2– остаток от деления а на 2 поместить в с (0 для четных, 1 для нечетных)

c = a % 10 – остаток от деления а на 10 поместить в с (последняя цифра числа)

c = a\*\*3 – возвести в третью степень

a = b = 5 – a и b получают одинаковое значение

a,b = 4,9 a = 4, b = 9

a = [1,2,3,4]

b,c,d,e = a

b=1, c=2, d=3, e=4

a == 5 – истина, если а равно 5, ложь в противном случае

a != 5 – истина, если а не равно 5, ложь в противном случае

not a == 5 – истина, если а не равно 5, ложь в противном случае

a == 5 or a == 10 – истина, если а равно 5 или 10, ложь в противном случае

a == 5 and b == 5 – истина, если а равно 5 и b равно 5, ложь в противном случае

a in range(10) – истина, если а >=0 и a<10

a in range(1,11,3) – истина, если а = 1 или 4 или 7 или 10

a in range(-10,10) – истина, если а >=-10 и a<10

not a in range(-10,10) – истина, если а <-10 или a>=10

a % 2 == 0 – истина, если а - четное

a % 3 != 0 – истина, если а не делится на 3

a % 10 == 3 – истина, если последняя цифра а – 3

Если программа не запускается

Возможные сообщения об ошибках и причины их появления:

|  |  |  |  |
| --- | --- | --- | --- |
| сообщение | смысловой перевод | код, вызвавший ошибку | возможное исправление |
| NameError: name 'b' is not defined | переменная ‘b’ не существует | **a = b +1** | b = 10 a = b + 1 |
| if a > 0  ^  SyntaxError: invalid syntax | синтаксическая ошибка (нет двоеточия) | if a > 0  print('+') | if a > 0:  print('+') |
| def test:  ^  SyntaxError: invalid syntax | синтаксическая ошибка (нет скобок) | def test:  print('test')  test() | def test():  print('test')  test() |
| нет сообщений, но не запускается функция | пропущены скобки | def test:  print('test')  test | def test:  print('test')  test() |
| print(a)  ^  IndentationError: expected an indented block | требуется отступ | a = 10  if a > 0:  print(a) | a = 10  if a > 0:  print(a) |
| UnboundLocalError: local variable 'color' referenced before assignment | переменная должна получить значение прежде, чем мы на это значение будем смотреть | def test():  while a < 10:  a += 1  test() | def test():  a = 1  while a < 10:  a += 1  test() |
| print('!')  ^  IndentationError: unindent does not match any outer indentation level | на одном уровне отступа разное количество символов отступа или смешаны типы отступов (в одной строке 4 пробела, а в другой – символ табуляции) | a = 10  if a > 0:  print(a)  print('!') | a = 10  if a > 0:  print(a)  print('!') |
| print(a)  ^  IndentationError: unexpected indent | отступ не нужен | a = 10  print(a) | a = 10  print(a) |
| print (a+b)  TypeError: unsupported operand type(s) for +: 'int' and 'str' | невозможно выполнить операцию «+» для числа и строки | a = 10  b = ‘10’  print (a+b) | a = ‘10’  b = ‘10’  print (a+b)  или  a = 10  b = 10  print (a+b) |
| a += 2  TypeError: 'int' object is not iterable | попытка добавить к списку целое число (нужно добавлять список, состоящий из одного значения) | a = [1,2,3,4]  a += a[0] | a = [1,2,3,4]  a += [a[0]]  или  a = [1,2,3,4]  a += a[:1] |
| test()  TypeError: test() takes exactly 1 argument (0 given) | функция не получила ожидаемого аргумента | def test(a):  print(‘!’)  test() | def test(a):  print(‘!’)  test(10)  или  def test():  print(‘!’)  test()  или  def test(a=10):  print(a)  test() |
| test(10)  TypeError: test() takes no arguments (1 given) | функция получила аргументы, который не ожидался | def test():  print(a)  test(10) | def test(a):  print(a)  test(10)  или  def test():  a = 10  print(a)  test() |
| print(a[4])  IndexError: list index out of range | попытка обратиться к элементу последовательности, которого не существует. | a=[1,2,3,4] print(a[4]) | a=[1,2,3,4] print(a[3]) |

команды черепашки

forward(10) - вперед на 10

fd(10) - вперед на 10

bk(100) - тоже что и fd(-100) – назад на 100

right(90) - тоже что и rt(90) , тоже что и lt(-90)– повернуть на 90 градусов направо (повернуть на месте)

lt(90) - тоже что и left(90), тоже, что и rt(-90) – повернуть на 90 градусов направо (повернуть на месте)

setpos(-100,100) – тоже же, что и goto(-100,100) -переместиться в точку -100,100

begin\_fill() - начать закрашенную фигуру

end\_gill() - закончить закрашенную фигуру

color = (‘red’,’blue’) изменить цвета линий и заливки

circle (40) – рисовать окружность радиусом 40

home() – вернуться в начальную точку в центре экрана, голова – направо

dot() – нарисовать точку

speed(1) – очень медленно

speed (6) – нормальная скорость

speed(10) - быстро

speed(0) – максимальная скорость

pd() =pendown() = down() – опустить перо (будет оставлять след)

pu() = penup() = up() – поднять перо (не будет оставлять след)

st() = showturtle() – показать черепашку

ht() = hideturtle() – спрятать черепашку

Логические операции

and, or, xor – бинарные логические операторы, применяются к логическим выражениям. Результат – логическое значение (Истина или Ложь)

not – унарный логический оператор

A,B – это логические выражения, например выражения, содержащие операторы отношений: (x < 10) или (x % 2 == 0), (10 in [1,2,3,4]) или др.

Для упрощения воспользуемся традиционной заменой: Истину обозначим как 1, а Ложь – как 0

|  |  |  |
| --- | --- | --- |
| A | B | A and B |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Это означает, что оператор И(and) возвращает истину только тогда, когда верны обе части.

|  |  |  |
| --- | --- | --- |
| A | B | A or B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Оператор ИЛИ (or) вернет Истину, если есть хотя бы одна верная часть.

|  |  |  |
| --- | --- | --- |
| A | B | A xor B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Оператор «исключающее ИЛИ» можно запомнить как «ИЛИ-ИЛИ»

|  |  |
| --- | --- |
| A | not A |
| 0 | 1 |
| 1 | 0 |

Отрицание просто меняет Истину на Ложь и наоборот.

Порядок вычисления: not, and, or, xor.

Примеры:

Будет напечатан «+»:

a = 10

b = 20

if a > 0 and b > 0 :

print(‘+’)

else:

print(‘-’)

Будет напечатан «-»:

a = 10

b = 2

if a > 0 and b > 0 :

print(‘+’)

else:

print(‘-’)

Будет напечатан «+» при любых значениях a,b:

a = -10

b = 20

if a > 0 and b > 0 or 1 != 2:

print(‘+’)

else:

print(‘-’)

(a > 10 and a < 20) дает точно такой же результат, как и not (a <= 10 or a >= 20)

Основные конструкции

Функция:

def test():

print(‘test’)

Вызов функции:

test()

Результат: на экране “test”

Аналогично, но функция не печатает, а возвращает значение (основной вариант)

def test():

return ‘test’

**Вызов ф**ункци**и:**

print(test())

Функция с параметром (получает один аргумент):

def test(a):

print(a+10)

Цикл for:

for x in [1,2,3,4,5,6]:

print(x)

x по очереди принимает значения из указанного списка

Цикл for + range:

for x in range(7):

print(x)

тоже, что и в предыдущем примере

Цикл for + range:

for x in range(10,100,10):

print(x)

x по очереди принимает значения от 10 до 90, с шагом 10. 100 не достигается!

Оператор выбора (ветвление), краткий вариант (без else):

if a > 10:

print(a)

Выводит на экран значение переменной, если она больше 10

Оператор выбора (полный вариант, c else)

if a > 10:

print(‘ a > 10’)

else:

print(‘a<=10’)

Выводит на экран строку a > 10 или a < 10, в зависимости от значения переменной

Оператор выбора:

if a == 10:

print(‘+’)

Оператор выбора:

if a != 10:

print(‘+’)

Оператор множественного выбора (несколько вариантов):

if a ==10:

print(‘+’)

elif a == 5:

print(‘++’)

elif a > 20:

print(‘a>20’)

else:

print(‘-’)

Оператор выбора + логические операции:

a = 10

b = 20

if a < b and a % 2 == 0:

print(‘+’)

Цикл while:

a = 0

while a < 10:

a += 1

print(a)

Выводит на экран числа от 1 до 10

Цикл while:

a = 10

while a >= 0:

a -= 1

print(a)

Выводит на экран числа от 9 до -1

Цикл while:

a = 0

while a < 0:

print(a)

Не выполнится ни разу. Условие ложно.

Цикл while:

a = 10

while a > 0:

a += 1

print(a)

Выполняется бесконечно, т.к. условие всегда истинно

Оператор break:

for x in [1,2,3,4]:

if x == 3:

break()

Прерывает работу цикла

Получение случайного числа:

import random

print(random.randrange(100))

Выводит на экран целое случайное число в отрезке от 0 до 99

или так:

from random import randrange

print (randrange(100))

или так:

from random import randrange as rnd

print (rnd(100))

Использование модулей:

import math

print (math.sin(10))

Выводит значение синуса 10 (угол в радианах!)

или так:

from math import \*

print (sin(10))

Типы

a = 5 – целое число (int)

a = 5.0 – вещественное число (float)

a = 10//4 - результат 2, целое число (int)

a = 10/4 = 2.5 (float)

a = ‘Hello’ – строка (str)

a = «Hello» - строка (str)

a = «555» - строка (str)

a = 5+5 – результат 10 (Int)

a = ‘5’+’5’ – результат ‘55’, строка (str)

a = “5”+”5” – то же, что в предыдущем случае

а = “5+5” – результат “5+5”, строка (str)

print(type(a)) – вывод названия типа

a = 10

print (type(a))

на экране int

a = ‘10’

print (type(a))

на экране str

a = ‘10’

print(type(int(a)))

на экране int

a = 10

a = str(a)

print (type(a))

на экране str

Списки

Функции, методы а также некоторые приемы работы со списками

a = [‘red’,’blue’,’yellow’,’green’,’black’,’white’]

a[0] – взять первое значение списка (‘red’)

a[1] – взять второе значение списка (‘blue’)

a[-1] – взять последнее значение (‘white’)

a[-2] - взять предпоследнее значение (‘black’)

a.pop() – удалить первый элемент списка

a.pop(0) – удалить последний элемент списка

a = a[2:] - взять часть списка, от ‘yellow’ и до конца

a = a[:2] – взять часть списка, от начала и до ‘blue’ включительно

a = a[2:5] – взять часть списка ‘yellow’,’green’,’black’

a = a[1:] – удалить первый

a = a[:-1] – удалить последний

a = a[:-3] – удалить последние три

a = a[::-1] – перевернуть список

a.reverse() – перевернуть список

a = a[::] - без изменений: копия списка

a = a[::2] - взять часть списка: элементы, стоящие на четных местах (0 тоже входит)

a = a[1::2] - взять часть списка: элементы, стоящие на нечетных местах

len(a) – получить длину списка (количество всех элементов)

sum(a) – получить сумму элементов списка (только для списков, содержащих числа)

if ‘brown’ in a: - проверить, входит ли элемент в список

a.index[0] – возвращает номер элемента, равного нулю. Если такого нет, то возникает ошибка

a.count(0) – возвращает количество элементов, равных нулю

a = a.sort() – возвращает сортированный список

a = a.sort(reverse=1) – возвращает сортированный список по убыванию

a = [0 for x in range(12)] – создать список из 12-ти элементов, равных нулю

a = [0]\*12 – тоже, что и выше

a = [x for x in range(12)] – создать список, содержащий 12 элементов: 1,2,3, … 11

a = [x for x in range(12) if x % 2 == 0] – создать список из четных значений от 0 до 11

a = [x for x in a if x % 2 == 0] – оставить в списке только четные элементы

a = [x for x in a if x != 0] – удалить все нули

a = [abs(x) for x in a] – все значения списка сделать положительными

Перебор по значениям. Элементы списка изменить таким образом невозможно.

for x in a:

print (x)

Вывести все значения списка.

Перебор по номерам. Используется, если нужно изменить элементы списка или получить их номера.

for i in range(len(a)):  
 a[i] += 10  
 print (a[i])

Увеличить значения списка и вывести

print(a)

Вывести все значения списка

print( ' '.join(map(str,a)))

Вывести все значения списка как одну строку, без запятых

a = [10/3,10/6,20/6,20/3,1/3]

for x in a:

print('%6.1f' % x, end = '')

print()

print (a)

Красиво вывести список, состоящий из дробных значений. Последняя строка дана для сравнения.

Строки

s = «Hello, world! I’am fine!»

s.index[‘ ‘] – вернет позицию пробела или ошибку, если пробела нет

s.find(‘ ’) – вернет позицию пробела или -1, если пробела нет

a = s.split(‘ ’) – вернет список, разделенный по пробелу: ['Hello,', 'world!', "I'am", 'fine!']

s = «Hello, world! \n I’am fine!» - при выводе строка будет разделена на две строки

s = 'Hello, world! I\'am fine!' – используется «\» для экранирования апострофа, чтобы вывести его как часть строки

s = s + chr(34) – добавление кавычек к строке

Tkinter

Cоздать окно:

from tkinter import \*

root = Tk()

mainloop()

Cоздать холст для рисования:

from tkinter import \*

root = Tk()

canv = Canvas(root)

canv.pack()

mainloop()

Создать кнопку и холст для рисования. Если поменять местами bt.pack() и canv.pack(), то кнопка будет под холстом.

from tkinter import \*

root = Tk()

bt = Button(root,text = 'Hello!')

canv = Canvas(root)

bt.pack()

canv.pack()

mainloop()

Две кнопки друг под другом. Размер окна – минимально возможный.

from tkinter import \*

root = Tk()

bt1 = Button(root,text = 'Hello 1')

bt1.pack()

bt2 = Button(root,text = 'Hello 2')

bt2.pack()

mainloop()

Создать окно определенного размера, две кнопки в первой строке

from tkinter import \*

root = Tk()

root.geometry('600x480')

fr = Frame(root)

bt1 = Button(fr,text = 'Hello 1')

bt2 = Button(fr,text = 'Hello 2')

bt1.pack(side='left',padx=2)

bt2.pack(side='left',padx=2)

fr.pack(pady=5)

mainloop()

Создать окно, две кнопки и холст для рисования

from tkinter import \*

root = Tk()

fr = Frame(root)

bt1 = Button(fr,text = 'Hello 1')

bt2 = Button(fr,text = 'Hello 2')

bt1.pack(side='left',padx=2)

bt2.pack(side='left',padx=2)

fr.pack(pady=5, fill = X)

canv = Canvas(root,width=800,height=600)

canv.pack()

mainloop()

При размещении с помощью pack можно использовать различные параметры:  
bt.pack(side = 'left') – прижать к левой стороне

bt.pack(side = 'left', padx = 5) – использовать отступ по горизонали (т.е. размещать не вполную к другим элементам этой строки)

bt.pack(side = 'left', pady = 5) – использовать отступ по вертикали (т.е. размещать не вполную к другим строкам)

bt.pack(fill = X) – игнорировать ширину виджета, растянуть так, чтобы занять все доступное пространство по ширине (незанятое другими виджетами)

expand=1 в сочетании с fill = BOTH заставит виджет занимать все доступное пространоство по ширине и высоте при изменении размеров окна

from tkinter import \*

root = Tk()

fr = Frame(root)

root.geometry('800x600')

bt1 = Button(fr,text = 'Hello 1')

bt2 = Button(fr,text = 'Hello 2')

bt1.pack(side='left',padx=2)

bt2.pack(side='left',padx=2)

fr.pack(pady=5, fill = X)

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

mainloop()

Button – кнопка

Text – многострочное текстовое поле для ввода

Entry –однострочное текстовое поле для ввода

Label – надпись

Canvas – холст для рисования

Эти и другие виджеты прекрасно описаны на сайте: http://effbot.org/tkinterbook/tkinter-index.htm#class-reference

bt.bind(‘<Button>’,test) – при щелчке любой кнопкой мыши на bt вызывать функцию test

canv.bind(‘<Motion>’,move) – при перемещении мыши на canv вызывать функцию move

Получить данные о событии:

from tkinter import \*

from random import randrange as rnd

root = Tk()

fr = Frame(root)

root.geometry('800x600')

bt1 = Button(fr,text = 'Hello 1')

bt2 = Button(fr,text = 'Hello 2')

bt1.pack(side='left',padx=2)

bt2.pack(side='left',padx=2)

fr.pack(pady=5, fill = X)

canv = Canvas(root, bg = 'white')

canv.pack(fill=BOTH,expand=1)

colors = ['orange','red','blue','yellow','green']

def click(event):

global color

color = colors[rnd(len(colors))]

color = colors[rnd(len(colors))]

def move(event):

x = event.x

y = event.y

canv.create\_oval(x-30,y-30,x+30,y+30,fill=color)

canv.bind('<Motion>',move)

canv.bind('<Button>',click)

mainloop()

(данный раздел следует дополнить: больше информации о виджетах, событиях, bind (третий параметр)))

Tkinter.Canvas

методы и константы, графические примитивы, сохранение изображения

canv = Canvas(root,width=800,height=600,bg=’white’) – создать холст (канвас) в окне root, указанными размерами и залить белым цветом

canv.create\_line(0,0,50,70) – создать линию из точки 0,0 в точку 50,70

canv.create\_line(0,0,50,70,width=4,fill='red') – толщина 4, цвет линии – красный

canv.create\_rectangle(10,30,50,80) – прямоугольник, левый верхний угол которого в точке 10,30, правый нижний – в точке 50,80

canv.create\_rectangle(10,30,50,80,fill = 'green') – зеленый прямоугольник

canv.create\_oval(10,30,50,80,fill = 'green', width=0) – зеленый круг без окантовки. Размеры определяются описанным прямоугольником

canv.create\_text(50,50,text=’Hello!’) – нарисовать текст c центром в указанной координате

canv.create\_text(50,50,text=’Hello!’,font = 'Tahoma 20 bold') – нарисовать текст указанным шрифтом

Чтобы можно было управлять элементом после создания, нужно каждому объекту давать имя:

a = canv.create\_line(0,0,50,70)  
canv.coords(a,(20,30,400,400))

Перемещает начало и конец линии в указанные точки

a = canv.create\_line(0,0,50,70)  
canv.itemconfig(a,fill=’green’,width=3)

Делает созданную линию зеленой и толстой

a = canv.create\_line(0,0,50,70)

canv.delete(a)

Удаляет линию

Удалить все объекты:

canv.delete(ALL)

Область видимости переменных (перенести выше)

a = 20

def test():

a = 10  
 print(a)  
print(a)

на экране будет 10 а потом 20

a = 20

def test():  
 b = 10  
 print(a)  
print(a)

На экране будет 20, потом 20

a = 20

def test():  
 global a

a = 10  
 print(a)  
print(a)

На экране будет 10, потом 10!

a = 20

def test():  
 b = 10  
 print(с)

Ошибка: переменная не определена

Правило такое: если внутри функции создается переменная, то внешняя переменная с таким же именем будет недоступна и любые изменения этой переменной останутся внутри функции. Если внутри функции нет переменной, то ищется глобальная переменная. В этом случае все изменения сохранятся и после завершения работы функции (плохой прием).

z = 20

def test (a,b):

return a+b

print (test(z,30))

Функция получает два аргумента и возвращает одно значение

z = 20

def test (a,b=10):

return a+b

print (test(z,30))  
print (test(z))

Функция получает один или два аргумента и возвращает одно значение. Если получен один только один аргумент, то второй считается равным 10.

z = 20

def test (a=20,b=10):

return a+b

print (test())  
print (test(10))  
print (test(10,20))

Значение “по умолчанию” может быть у всех аргументов.

def test (a,b):

if a > b:

return (a,b)

else:

return (b,a)

print (test(30,20))

Функция получает два аргумента и возвращает последовательность (кортеж), содержащий два значения. Так можно возвращать более одного значения.

Указатели

a = 10

b = a

b += 20

print (a,b)

На экране 10, 30

a = [10]

b = a

b += [20]

print (a,b)

На экране [10,20], [10,20].

Как ни странно, но список а также изменился, несмотря на то, что мы его, вроде бы, не изменяли.

Для чисел, строк и переменных True/False работает первый вариант, для всех остальных – второй.

При копировании объектов (т.е. все, кроме строк, чисел и True/False) происходит не копирование значения, а создание нового имени для того же объекта!

a = [10,20]

def test (a):

a += ['!']

return a

print (a)

print (test(a))

print (a)

На экране:  
[10, 20]  
[10, 20, '!']  
[10, 20, '!']  
Т.е. изменения списка а сохранились после завершения работы функции.

Если нужно передать копию списка, тогда так:

a = [10,20]

def test (a):

a += ['!']

return a

print (a)

print (test(a[::]))

print (a)