Программирование

PYTHON

Тема1: Введение.

PYTHON

Тема1: Введение.

https://docs.python.org/3/index.html
https://www.programiz.com/python-programming/tutorial
https://pythoner.name/
https://metanit.com/python/tutorial/
https://python-scripts.com/
https://pythonz.net/references/named/python/
https://pythonworld.ru/samouchitel-python

PYTHON

Тема1: Введение.

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Эталонной реализацией Python является интерпретатор Cpython.

PYTHON

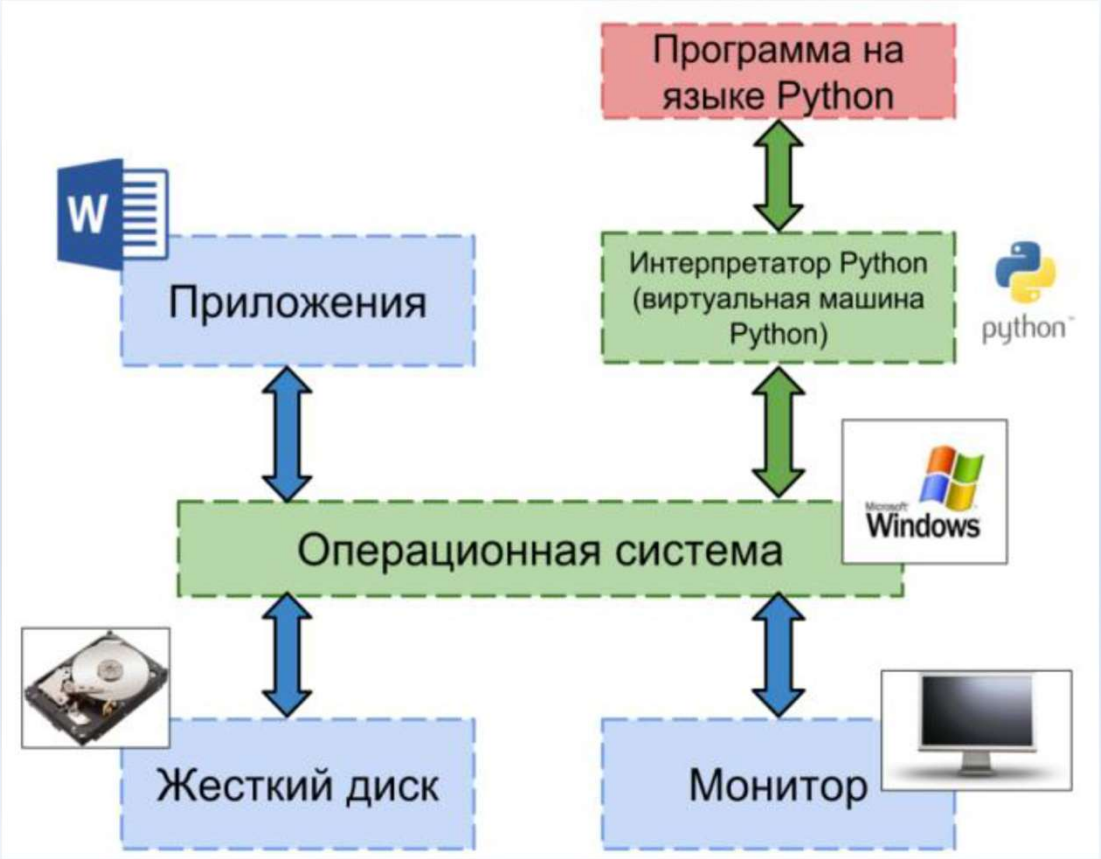
Тема1: Введение.

ХНЭУ, факультет ЭИ, кафедра ИС,
тел. 702-18-31, e-mail: volodymyr.fedorchenko@hneu.net

доцент кафедры информационных систем
Федорченко Владимир Николаевич

PYTHON

Тема1: Введение.



ХНЭУ, факультет ЭИ, кафедра ИС,
тел. 702-18-31, e-mail: volodymyr.fedorchenko@hneu.net

доцент кафедры информационных систем
Федорченко Владимир Николаевич

PYTHON

Тема1: Введение.

Implementation	Virtual Machine	Ex) Compatible Language
CPython	CPython VM	C
Jython	JVM	Java
IronPython	CLR	C#
Brython	Javascript engine (e.g., V8)	JavaScript
RubyPython	Ruby VM	Ruby

PYTHON

Тема1: Введение.

PyPy
Cython
Numba
Ipython
PyObjc
PyQt
pyjs (Pyjamas)-Brython

PYTHON

Тема1: Введение.

https://www.python.org/

PYTHON

Тема1: Введение.

Существует несколько специализированных IDE для разработки на Python.

•PyCharm от JetBrains — *платформы* Windows, Mac OS X и Linux, *поддержка* Community и Professional вариантах.

•Wing IDE

«Wing 101» -- free

«Wing Personal» -- free

«Wing Pro – платная:

поддержка проектов

работу с SCV (системами управления версиями)

расширенные возможности навигации по коду и анализа кода

рефакторинг

поддержка использования Django

PYTHON

Тема1: Введение.

- Spyder
  - open-source IDE для Python под лицензией MIT, бесплатная,
  - for Windows, Mac OS X and Linux.
  - specified for data science, fremwork SciPy, NumPy, Matplotlib.
  - packet manager Anaconda.

Plugin Python for IDE:

PyDev eclipse base

Microsoft Visual Studio

а также имеется поддержка подсветки синтаксиса, автодополнения кода и подключения средств отладки и запуска приложений для целого ряда распространённых текстовых редакторов.

AP1

PYTHON

Тема1: Введение.

- ***Anaconda - IPython и Jupyter Notebook***

Slide 11

AP1 Andrii Poliakov, 1/15/2020

Тема1: Введение в написание программ.

Программа на языке Python состоит из набора инструкций. Каждая инструкция помещается на новую строку. Например:

```
print(2 + 3)
print("Hello")
```

Большую роль в Python играют отступы. Неправильно поставленный отступ фактически является ошибкой. Например, в следующем случае мы получим ошибку, хотя код будет практически аналогичен приведенному выше:

```
1     print(2 + 3)
2         print("Hello")
```

Поэтому стоит помещать новые инструкции сначала строки. В этом одно из важных отличий пайтона от других языков программирования, как C# или Java.

Однако стоит учитывать, что некоторые конструкции языка могут состоять из нескольких строк. Например, условная конструкция **if**:

```
1     if 1 < 2:
2         print("Hello")
```

В данном случае если 1 меньше 2, то выводится строка "Hello". И здесь уже должен быть отступ, так как инструкция print("Hello") используется не сама по себе, а как часть условной конструкции if. Причем отступ, согласно руководству по оформлению кода, желательно делать из такого количества пробелов, которое кратно 4 (то есть 4, 8, 16 и т.д.

PYTHON

Тема1: Введение в написание программ.

Комментарии

Для отметки, что делает тот или иной участок кода, применяются комментарии. При трансляции и выполнении программы интерпретатор игнорирует комментарии, поэтому они не оказывают никакого влияния на работу программы.

Комментарии в Python бывают блочные и строчные. Все они предваряются знаком решетки (#).

Блочные комментарии ставятся в начале строки:

```
1    # Вывод сообщения на консоль
2    print("Hello World")
```

Строчные комментарии располагаются на той же строке, что и инструкции языка:

```
1    print("Hello World")  # Вывод сообщения на консоль
```

Shebang

Хорошей практикой программирования является размещения на первой строке специальной инструкции - шебанга (shebang), который представляет решетки с восклицательным знаком - своего рода особого комментария с указанием того, что делает данный скрипт.

```
1    #! Первая программа на Python
2
3    print("Hello World")  # Вывод сообщения на консоль
```

PYTHON

Тема1: Введение в написание программ.

Синтаксис

- Конец строки является концом инструкции (точка с запятой не требуется).
- Вложенные инструкции объединяются в блоки по величине отступов. Отступ может быть любым, главное, чтобы в пределах одного вложенного блока отступ был одинаков. И про читаемость кода не забывайте. Отступ в 1 пробел, к примеру, не лучшее решение. Используйте 4 пробела (или знак табуляции, на худой конец).
- Вложенные инструкции в Python записываются в соответствии с одним и тем же шаблоном, когда основная инструкция завершается двоеточием, вслед за которым располагается вложенный блок кода, обычно с отступом под строкой основной инструкции.

Основная инструкция:

Вложенный блок инструкций

Несколько специальных случаев

- Иногда возможно записать несколько инструкций в одной строке, разделяя их точкой с запятой:

```
a = 1; b = 2; print(a, b)
```

- Допустимо записывать одну инструкцию в нескольких строках. Достаточно ее заключить в пару круглых, квадратных или фигурных скобок:

```
if (a == 1 and b == 2 and
        c == 3 and d == 4):  # Не забываем про двоеточие
    print('spam' * 3)
```

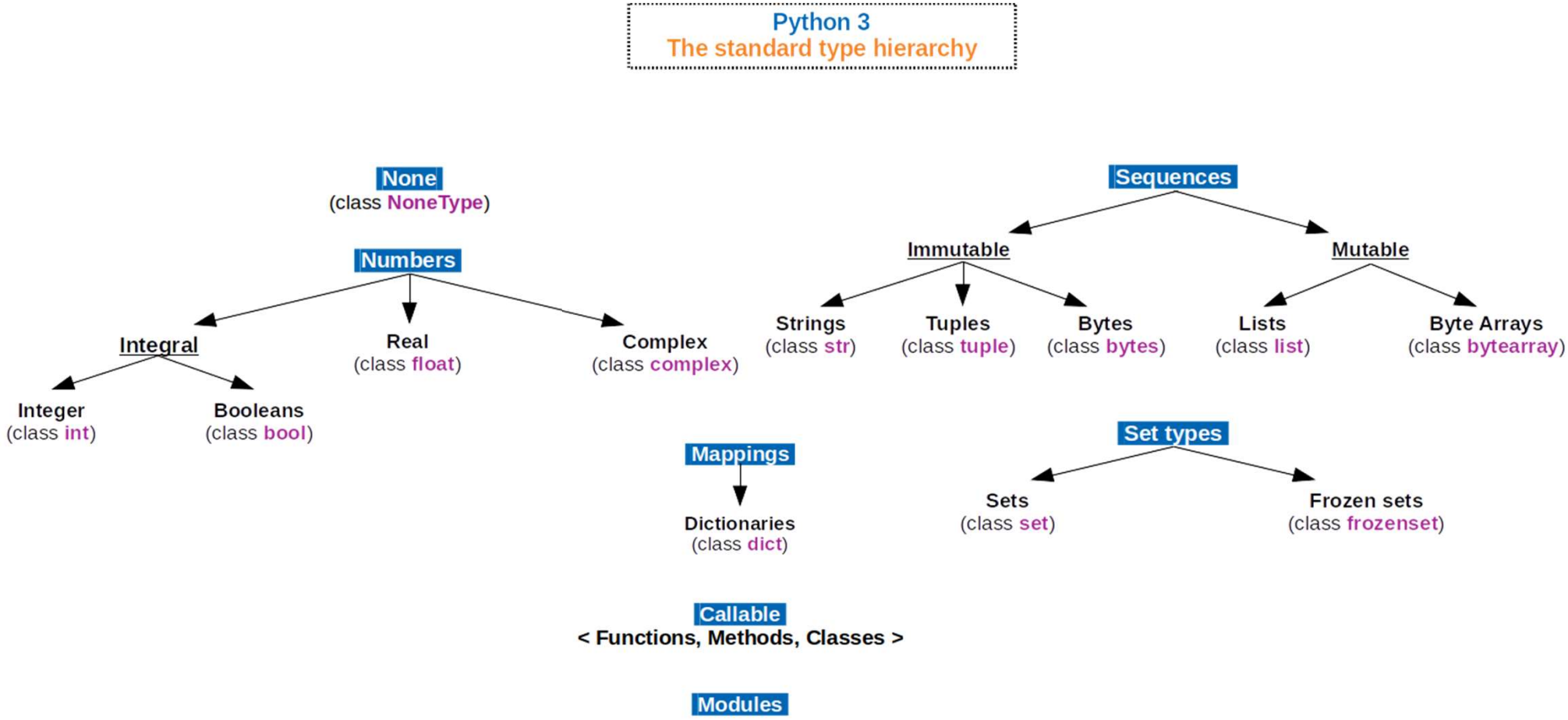
- Тело составной инструкции может располагаться в той же строке, что и тело основной, если тело составной инструкции не содержит составных инструкций.. Пример:

```
if x > y: print(x)
```

PYTHON

Тема1: Типы и модель данных.

Python относится к языкам с ***неявной сильной динамической типизацией***.



Типы данных в Python

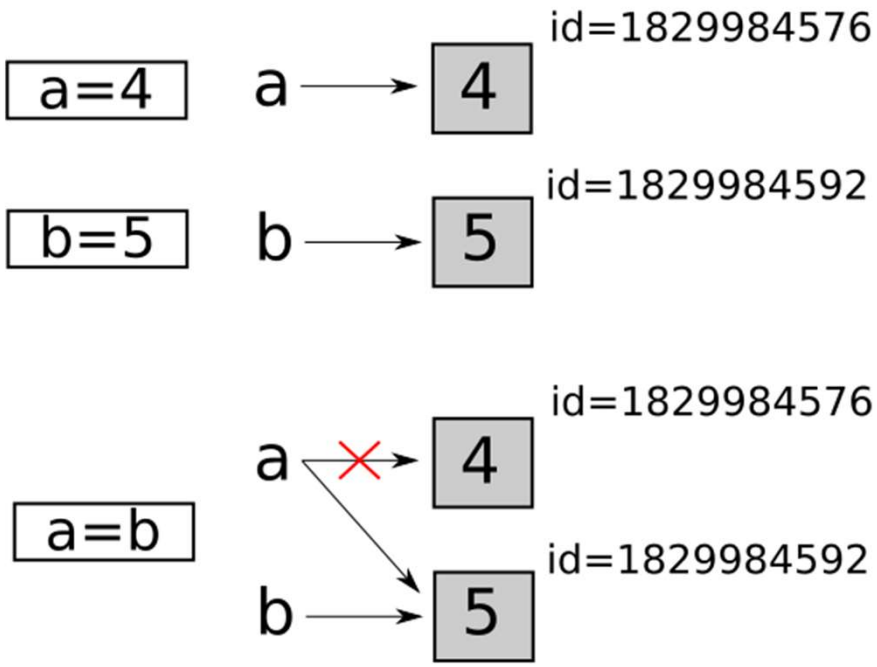
- В Python типы данных можно разделить на встроенные в интерпретатор (built-in) и не встроенные, которые можно использовать при импортировании соответствующих модулей.

К основным встроенным типам относятся:

- *None* (неопределенное значение переменной)
- Логические переменные (*Boolean Type*)
- Числа (*Numeric Type*)
  - *int* – целое число
  - *float* – число с плавающей точкой
  - *complex* – комплексное число
- Списки (*Sequence Type*)
  - *list* – список
  - *tuple* – кортеж
  - *range* – диапазон
- Строки (*Text Sequence Type*)
  - *str*
- Бинарные списки (*Binary Sequence Types*)
  - *bytes* – байты
  - *bytearray* – массивы байт
  - *memoryview* – специальные объекты для доступа к внутренним данным объекта через protocol buffer
- Множества (*Set Types*)
  - *set* – множество
  - *frozenset* – неизменяемое множество
- Словари (*Mapping Types*)
  - *dict* – словарь

PYTHON

Тема1: Типы и модель данных.



id(a)
1829984576

id(b)
1829984592

id(b)
1829984576

id(b)
1829984592

```
a = 4
b = 5
id(a)
id(b)
a = b
id(a)
```

PYTHON

Тема1: Типы и модель данных.

Тип переменной можно определить с помощью функции *type()*. Пример использования приведен ниже.

```
>>> a = 10
>>> b = "hello"
>>> c = (1, 2)
>>> type(a)
<class 'int’>
>>> type(b)
<class 'str’>
>>> type(c)
<class 'tuple'>
```

PYTHON

Изменяемые и неизменяемые типы данных

К **неизменяемым** (*immutable*) типам относятся: целые числа (*int*), числа с плавающей точкой (*float*), комплексные числа (*complex*), логические переменные (*bool*), кортежи (*tuple*), строки (*str*) и неизменяемые множества (*frozen set*).
К **изменяемым** (*mutable*) типам относятся: списки (*list*), множества (*set*), словари (*dict*).

При создании переменной, вначале создается объект, который имеет уникальный идентификатор, тип и значение, после этого переменная может ссылаться на созданный объект.

Неизменяемость типа данных означает, что созданный объект больше не изменяется. Например, если мы объявим переменную *k* = 15, то будет создан объект со значением 15, типа *int* и идентификатором, который можно узнать с помощью функции *id()*.

```
>>> k = 15
>>> id(k) 1672501744
>>> type(k)
<class 'int'>
```

Объект с *id* = 1672501744 будет иметь значение 15 и изменить его уже нельзя.

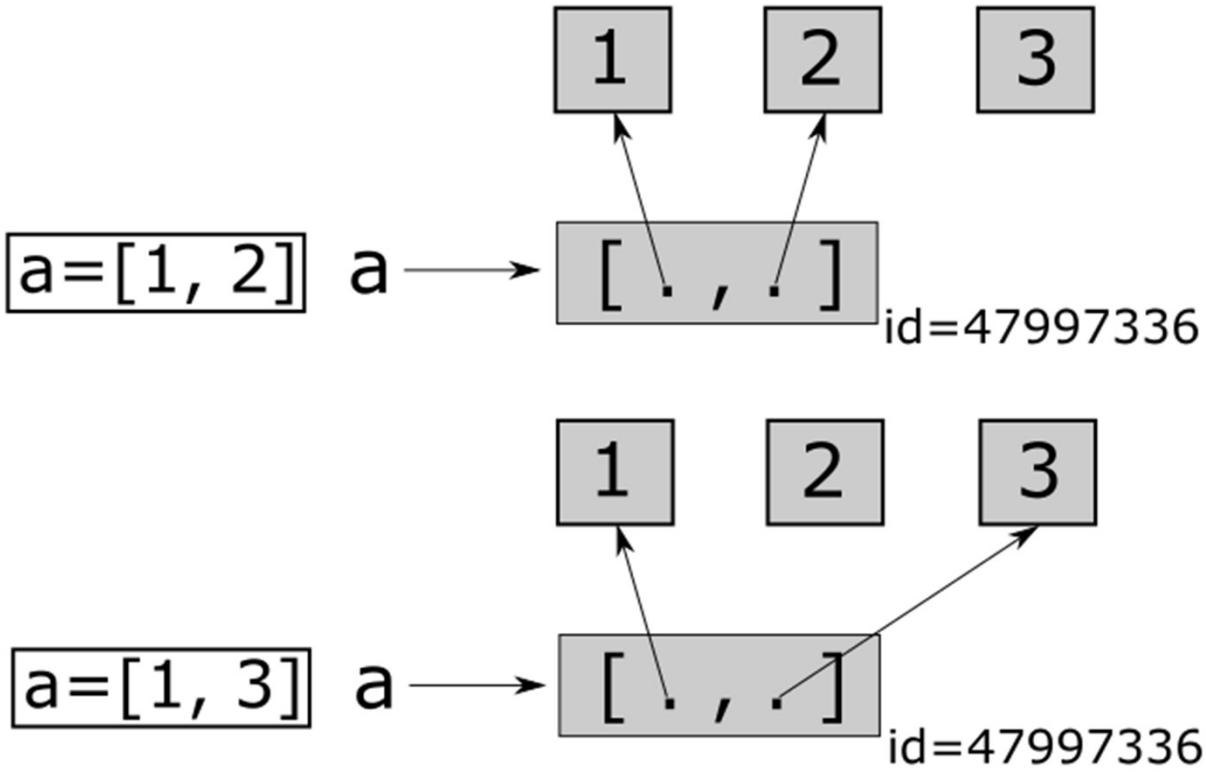
PYTHON

Тема1: Типы и модель данных: *type()*

Если тип данных **изменяемый**, то можно менять значение объекта.

Например, создадим список [1, 2], а потом заменим второй элемент на 3.

```
>>> a = [1, 2]
>>> id(a)
47997336
>>> a[1] = 3
>>> a [1, 3]
>>> id(a)
47997336
```



PYTHON

Тема1: Операции.

Оператор	Описание
**	Возведение в степень
~ + -	Комплиментарный оператор
* / % //	Умножение, деление, деление по модулю, целочисленное деление.
+ -	Сложение и вычитание.
>> <<	Побитовый сдвиг вправо и побитовый сдвиг влево.
&	Бинарный "И".
^ |	Бинарный "Исключительное ИЛИ" и бинарный "ИЛИ"
<= < > >=	Операторы сравнения
<> == !=	Операторы равенства
= %= /= //= -= += *= **=	Операторы присваивания
is is not	Тождественные операторы
in not in	Операторы членства
not or and	Логические операторы

PYTHON

Тема1: Операции.

Помимо стандартных выражений для работы с числами (а в Python их не так уж и много), в составе Python есть несколько полезных модулей.

Модуль math предоставляет более сложные математические функции.

```
>>>
>>> import math
>>> math.pi
3.141592653589793
>>> math.sqrt(85)
9.219544457292887
```

Модуль random реализует генератор случайных чисел и функции случайного выбора.

```
>>>
>>> import random
>>> random.random()
0.15651968855132303
```

PYTHON

Тема1: Системы счисления.

- **int**([object], [основание системы счисления]) - преобразование к целому числу в десятичной системе счисления. По умолчанию система счисления десятичная, но можно задать любое основание от 2 до 36 включительно.
- **bin**(x) - преобразование целого числа в двоичную строку.
- **hex**(x) - преобразование целого числа в шестнадцатеричную строку.
- **oct**(x) - преобразование целого числа в восьмеричную строку.

Представление числа

При обычном определении числовой переменной она получает значение в десятичной системе. Но кроме десятичной в Python мы можем использовать двоичную, восьмеричную и шестнадцатеричную системы.

Для определения числа в двоичной системе перед его значением ставится **0** и префикс b:

```
x = 0b101           # 101 в двоичной системе равно 5
```

Для определения числа в восьмеричной системе перед его значением ставится **0** и префикс o:

```
a = 0o11            # 11 в восьмеричной системе равно 9
```

Для определения числа в шестнадцатеричной системе перед его значением ставится **0** и префикс x:

```
y = 0x0a            # a в шестнадцатеричной системе равно 10
```

И с числами в других системах измерения также можно проводить арифметические операции:

```
x = 0b101           # 5
y = 0x0a            # 10
z = x + y           # 15
print("{0} in binary {0:08b}   in hex {0:02x} in octal {0:02o}".format(z))
```

PYTHON

Тема1: Условная конструкция if.

```
if <boolean_expression>:
    инструкции
[elif <boolean_expression>:
    инструкции]
[else:
    инструкции]
```

Проверка истинности в Python

- Любое число, не равное 0, или непустой объект - истина.
- Числа, равные 0, пустые объекты и значение None - ложь
- Операции сравнения применяются к структурам данных рекурсивно
- Операции сравнения возвращают True или False
- Логические операторы and и or возвращают истинный или ложный объект-операнд

Трехместное выражение if/else

Следующая инструкция:

```
if X:
    A = Y
else:
    A = Z
```

довольно короткая, но, тем не менее, занимает целых 4 строки. Специально для таких случаев и было придумано выражение if/else:

```
A = Y if X else Z
```

В данной инструкции интерпретатор выполнит выражение Y, если X истинно, в противном случае выполнится выражение Z.

PYTHON

Тема1: Циклы.

Циклы позволяют повторять некоторое действие в зависимости от соблюдения некоторого условия.

Цикл while

while <boolean_expression>**:**
  инструкции

```
#!/usr/bin/env python

number = int(input("Введите число: "))
i = 1
factorial = 1
while i <= number:
    factorial *= i
    i += 1
print("Факториал числа", number, "равен", factorial)
```

PYTHON

Тема1: Циклы.

Циклы позволяют повторять некоторое действие в зависимости от соблюдения некоторого условия.

Цикл for

for <*int_var*> **in** <function_range>**:**
    <statements>

Полная форма цикла for имеет следующий вид:

```
for <target> in <object>:      # Присваивает элементы объекта с
                               # переменной цикла
   <statements>
   if <test>: break            # Выход из цикла, минуя блок else
   if <test>: continue         # Переход в начало цикла
else:
   <statements>                # Если не была вызвана инструкция 'break'
```

```
for i in 'hello world':
    print(i * 2, end='')
```

hheellllloo wwoorrlldd

PYTHON

Тема1: Циклы.

Циклы позволяют повторять некоторое действие в зависимости от соблюдения некоторого условия.

```
#!/usr/bin/env python

number = int(input("Введите число: "))
factorial = 1
for i in range(1, number+1):
    factorial *= i
print("Факториал числа", number, "равен", factorial)
```

PYTHON

Тема1: Циклы.

Циклы позволяют повторять некоторое действие в зависимости от соблюдения некоторого условия.

Цикл for - проще и быстрее чем цикл **while**. Но цикл for не всегда сможет сделать обход каких то элементов (к примеру обход первого или же второго элемента в каком то типе данных).
С такой задачей легко справляется цикл **while**. Но и для цикла for есть вспомогательные функции позволяющие управлять обходом элементов.

Встроенная функция **range** возвращает непрерывную последовательность увеличивающихся целых чисел, которые можно использовать в качестве индексов внутри цикла for.

Встроенная функция **zip** возвращает список кортежей, составленных из элементов входных списков с одинаковыми индексами, который может использоваться для одновременного обхода нескольких последовательностей в цикле for**.**

PYTHON

Тема1: Циклы.

Циклы позволяют повторять некоторое действие в зависимости от соблюдения некоторого условия.

Функция range

Функция range имеет следующие формы:

range(stop): возвращает все целые числа от 0 до stop (Возвращает пустую последовательность, если стоп отрицательный или 0)

range(start, stop): возвращает все целые числа в промежутке от start (включая) до stop (не включая). Выше в программе факториала использована именно эта форма.

range(start, stop, step): возвращает целые числа в промежутке от start (включая) до stop (не включая), которые увеличиваются на значение step

Примеры вызовов функции range:

```
range(5)          # 0, 1, 2, 3, 4
range(1, 5)       # 1, 2, 3, 4
range(2, 10, 2)   # 2, 4, 6, 8
range(5, 0, -1)   # 5, 4, 3, 2, 1
```

https://docs.python.org/3/library/stdtypes.html?highlight=range#range

PYTHON

Тема1: Циклы.

Циклы позволяют повторять некоторое действие в зависимости от соблюдения некоторого условия.

Вложенные циклы

Одни циклы внутри себя могут содержать другие циклы. Рассмотрим на примере вывода таблицы умножения:

```
for i in range(1, 10):
    for j in range(1, 10):
        print(i * j, end="\t")
    print("\n")
```

PYTHON

Тема1: Циклы.

Выход из цикла. break и continue

```
# !/usr/bin/env python

print("Для выхода нажмите Y")
while True:
    data = input("Введите сумму для обмена: ")
    if data.lower() == "y":
        break  # выход из цикла
    money = int(data)
    if money < 0:
        print("Сумма должна быть положительной!")
        continue
    cache = round(money / 56, 2)
    print("К выдаче", cache, "долларов")

print("Работа обменного пункта завершена")
```

PYTHON

Тема1: Циклы.

else:
pass
Слово else, примененное в цикле **for** или **while**, проверяет, был ли произведен выход из цикла инструкцией **break**, или же "естественным" образом. Блок инструкций внутри else выполнится только в том случае, если выход из цикла произошел без помощи **break**.

```
#!/usr/bin/env python

y = int(input("input number: "))

x = y // 2                          # Для значений y > 1
while x > 1:
    if y % x == 0:                  # Остаток
        print(y,'has factor', x)
        break                       # Перешагнуть блок else
    x -= 1
else:                               # Нормальное завершение цикла
    print(y, 'is prime')
pass
```

PYTHON

Тема1: Циклы.

```
# Program to check Armstrong numbers in certain interval
lower = 100
upper = 2000
# To take input from the user
# lower = int(input("Enter lower range: "))
# upper = int(input("Enter upper range: "))
for num in range(lower, upper + 1):   # order of number
   order = len(str(num))
   # initialize sum
   sum = 0
   # find the sum of the cube of each digit
   temp = num
   while temp > 0:
       digit = temp % 10
       sum += digit ** order
       temp //= 10
   if num == sum:
       print(num)
```

PYTHON

Тема2: Функции.

Функции представляют блок кода, который выполняет определенную задачу и который можно повторно использовать в других частях программы

```
def <function_name> ([parameters]):
    <statement>
```

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

say_hello()
say_hello()
say_hello()
```

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

say_hello()
say_hello()
say_hello()
```

PYTHON

Тема2: Функции.

Функции представляют блок кода, который выполняет определенную задачу и который можно повторно использовать в других частях программы

Вызывая функцию, мы можем передавать ей следующие типы аргументов:

- *Обязательные аргументы (Required arguments)*
- *Аргументы-ключевые слова (Keyword argument)*
- *Аргументы по умолчанию (Default argument)*
- *Аргументы произвольной длины (Variable-length argumens)*

Значения по умолчанию

Некоторые параметры функции мы можем сделать необязательными, указав для них значения по умолчанию при определении функции.

```
def say_hello(name="Tom"):
    print("Hello,", name)

say_hello()
say_hello("Bob")
```

Именованные параметры

При передаче значений функция сопоставляет их с параметрами в том порядке, в котором они передаются.

```
def display_info(name, age):
    print("Name:", name, "\t", "Age:", age)

display_info("Tom", 22)
```

```
def display_info(name, age):
    print("Name:", name, "\t", "Age:", age)

display_info(age=22, name="Tom")
```

PYTHON

Тема2: Функции.

Переменное количество позиционных аргументов.

```
>>> def func(*args):
... return args
...
>>> func(1, 2, 3, 'abc')
(1, 2, 3, 'abc')
>>> func()
()
>>> func(1) (1,)
```

PYTHON

Тема2: Функции.

Произвольное число именованных аргументов.

```
def func(**kwargs):
    return kwargs

func(a=1, b=2, c=3)
func()
func(a='python')
```