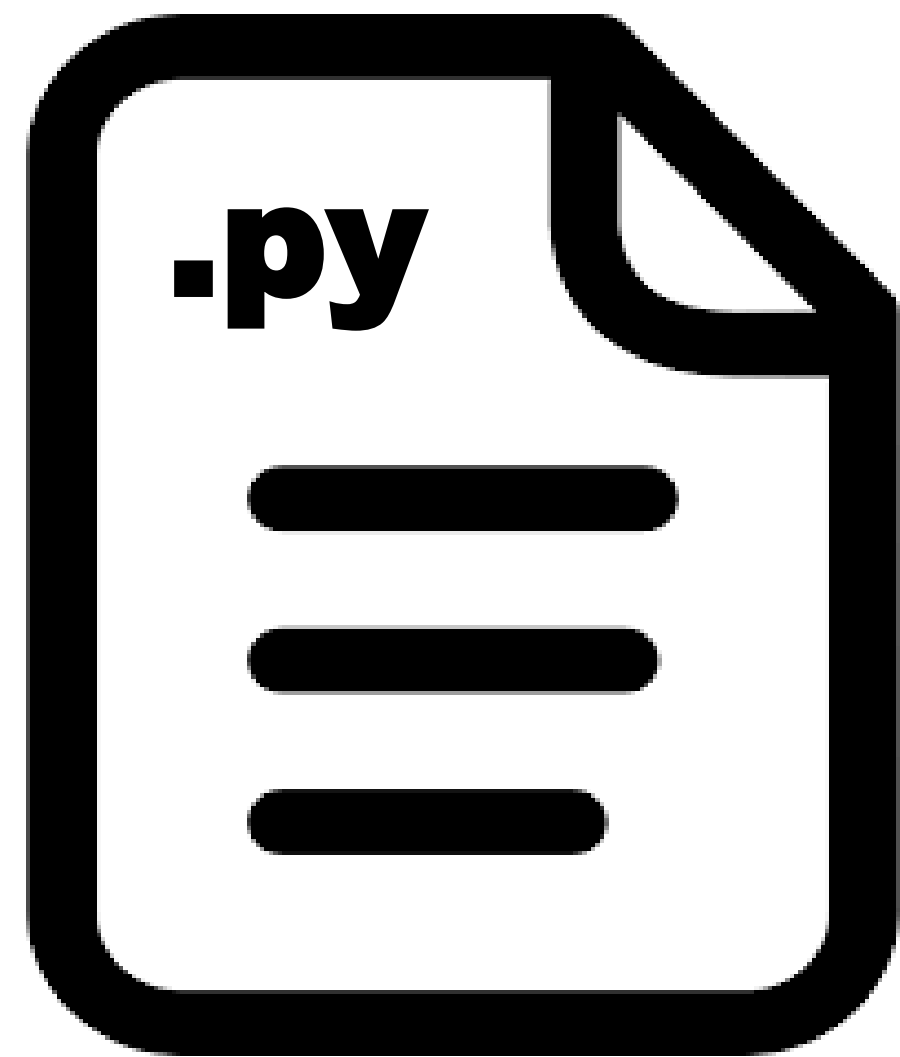


Структура языка Логический тип данных

Докладчик: Евграфов Михаил

Структура языка

Py-файл как множество строк



физические строки

```
light_velocity = 3e8
```

логические строки

```
numbers = [  
    1, 2, 3, 4,  
    5, 6, 7, 8,  
]
```

Комментарии

правильное использование комментариев

```
# максимальная дальность обзора РЛС в морских милях  
# согласно протоколу использования РЛС, с.3 таб.1  
distance_max = 72 * 1852
```

неправильное использование комментариев

```
distance_max = 72 * 1852 # определили переменную
```

Пустые строки

без использования пустых строк

```
if number % 2 == 0:  
    number_type = "even"  
else:  
    number_type = "odd"
```

```
if number % 2 == 0:  
    number_type = "even"  
  
else:  
    number_type = "odd"
```

с использованием пустых строк

Объединение физических строк

логическая строка

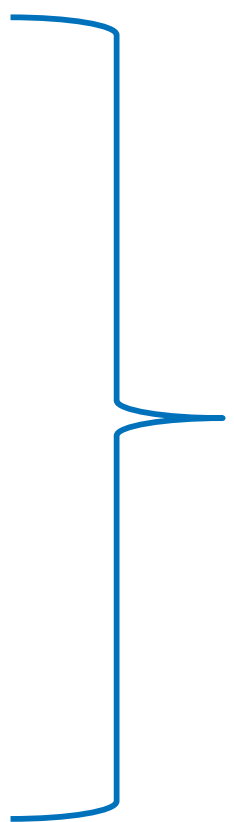
```
very_long_string = "this is long string so long " \
"so you need to postpone its part to the next line"
```

python-way логическая строка

```
another_long_string = (  
    "this is very very very long string so long "  
    "so you need to postpone its part"  
)
```

Отступы

```
while num > 0:
    binary_notation = (
        str(num % 2) + binary_notation
    )
    num //= 2
```



блок кода

```
if not binary_notation:
    binary_notation = "0"
```

Простые инструкции

операция привязки переменной - простая инструкция

```
light_velocity = 3e8
```

вызов функции - простая инструкция

```
print("Hello, world!")
```

использование оператора pass - простая инструкция

```
pass
```


Составные инструкции

первое положение сложной инструкции

if number % 2 == 0: # хедер первого положения

 # тело первого положения

 number_type = "even"

второе положение сложной инструкции

else: # хедер второго положения

 # тело второго положения

 number_type = "odd"

Токены

Виды токенов:

- **идентификаторы**
- **разделители**
- **литералы**
- **операторы**
- **ключевые слова**

Идентификаторы

```
my_variable = 5  # валидное имя
MyVariable = 6  # тоже валидное имя
_var1 = 1  # это тоже валидное имя
__len__ = 4  # и это валидное имя переменной
_2 = 2  # и это тоже

2var = 2  # SyntaxError
my_$uper_variable = 4  # SyntaxError
```


Разделители

()	[]	{	}	
,	:	.	;	@	=	->
+=	-=	*=	/=	//=	%=	@=
&=	=	^=	>>=	<<=	**=	

Литералы

1

целое число

3.14

число с плавающей точкой

1.0j

комплексное число

"Hello!"

строковый литерал

Операторы

+

-

*

**

/

//

%

@

<<

>>

&

|

^

~

:=

<

>

<=

>=

==

!=

Ключевые слова

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

Объекты Python

Характеристики объекта Python:

- **идентичность**
- **тип данных**
- **значение**

Идентичность

получение идентичности объекта

```
>>> id(5)  
140723671524264
```

сравнение идентичностей объектов

```
>>> print() is None
```

```
True
```


Тип данных

определение типа данных объекта

```
>>> type(None)
<class 'NoneType'>

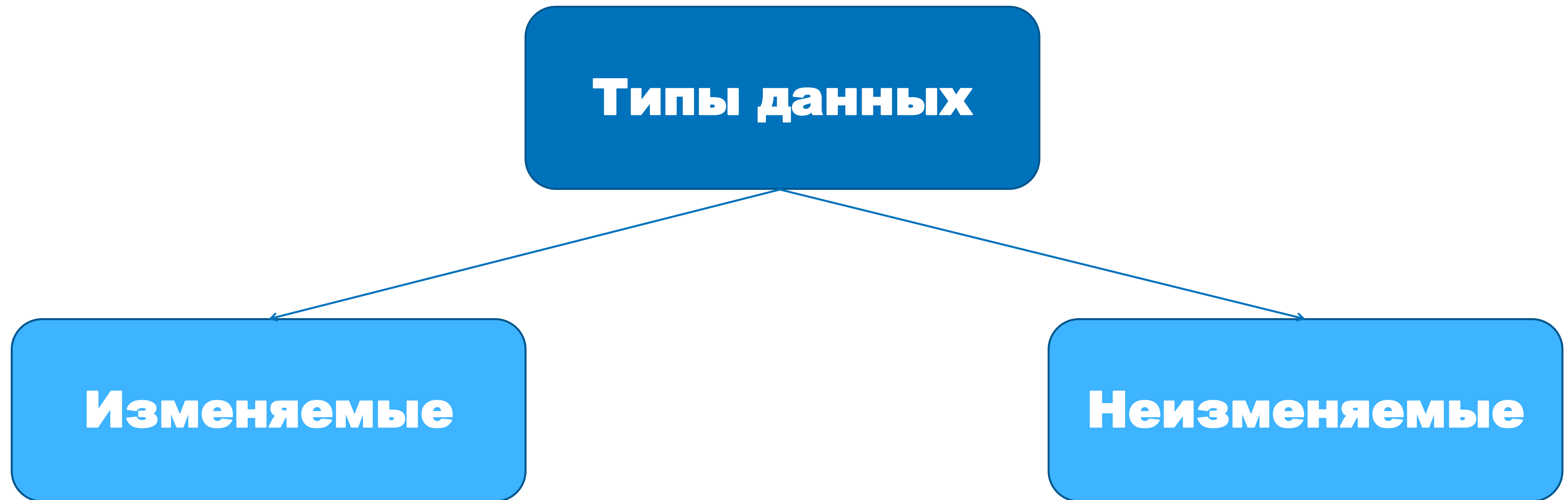
>>> type(None).__name__
'NoneType'
```

проверка типа данных объекта

```
>>> bool_flag = True
>>> type(bool_flag) is bool
True
```

```
>>> bool_flag = True
>>> isinstance(bool_flag, bool)
True
```

Типы данных



Переменные и ссылки

num1 = 5

num1



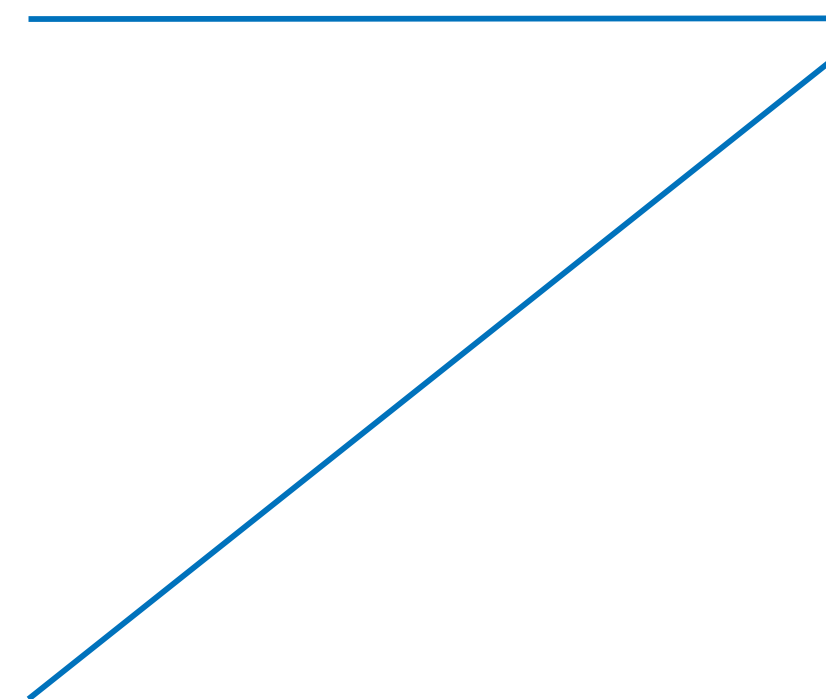
num1 = 5

num2 = num1

num1 = 6

num2 = num1

num2



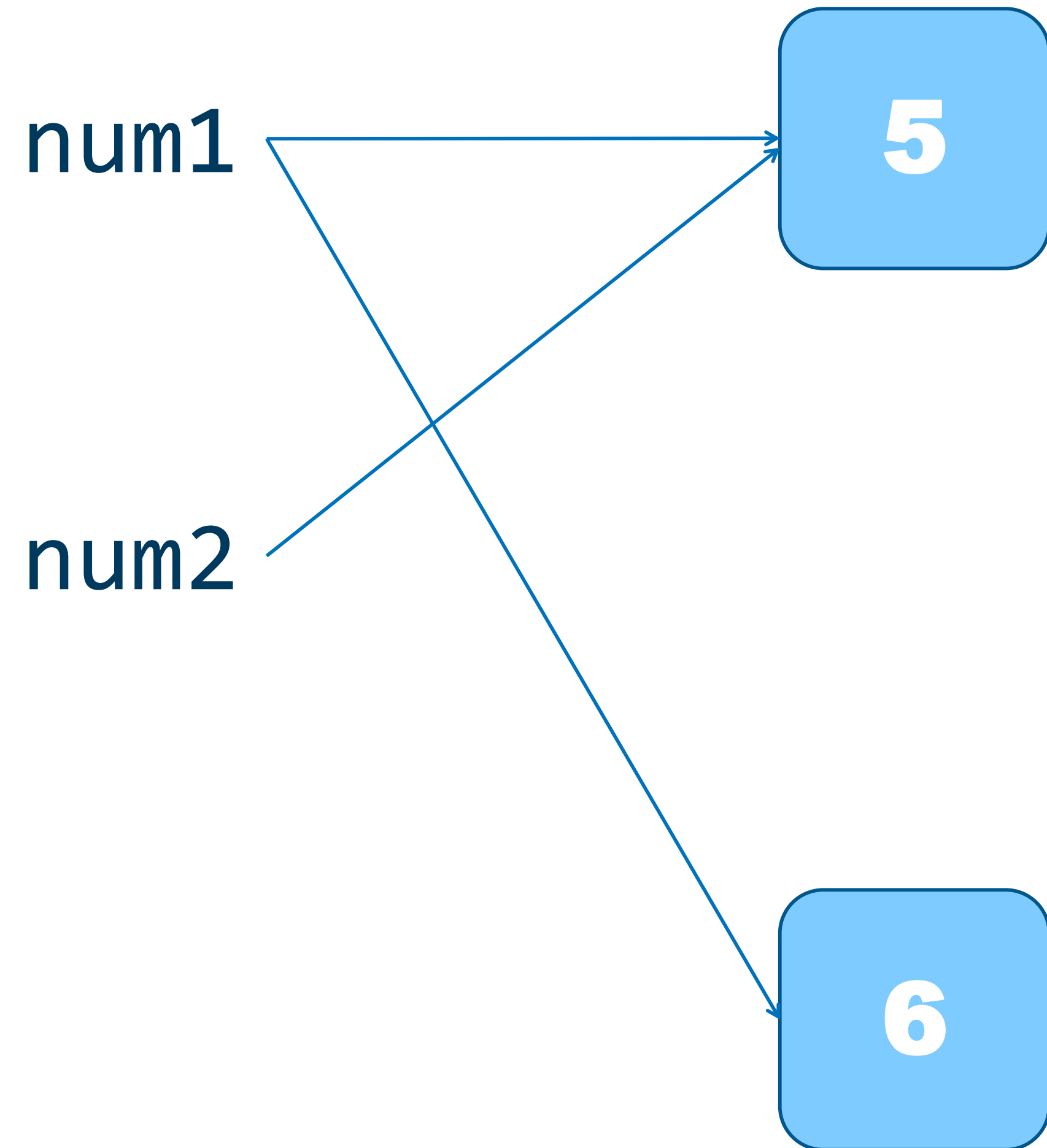
Переменные и ссылки

```
num1 = 5  
num2 = num1  
num1 = 6
```

num1 = 5

num2 = num1

num1 = 6



Виды присваивания

каскадное присваивание

```
num1 = num2 = num3 = 0
```

множественное присваивание

```
num1, num2 = 3, 4
```

```
num1, num2 = num2, num1
```


Логический тип данных

Основные сведения

- **логический тип - неизменяемый тип данных**
- **допустимые значения: `True` и `False`**
- **функция для конвертации: `bool()`**

Конвертация объектов в bool

```
>>> bool(0)  
False
```

```
>>> bool(3)  
True
```

```
>>> bool(-3)  
True
```

```
>>> bool(0.5)  
True
```

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

```
>>> bool([])  
False
```

```
>>> bool(None)  
False
```


Арифметические операции

унарный минус

-True # == -1

сложение

True + True # == 2

вычитание

True - False # == 1

умножение

True * True # == 1

честное деление (truediv)

True / True # == 1.0

целочисленное деление

True // True # == 1

модульное деление

True % True # == 0

Битовые операции

битовое НЕ

`~True` # == -2

битовое И

`True & False` # == False

битовое ИЛИ

`True | False` # == True

исключающее ИЛИ

`True ^ False` # == True

битовый сдвиг влево

`True << True` # == 2

битовый сдвиг вправо

`True >> True` # == 0

Логические операции

оператор равенства

True == False # == False

оператор неравенства

True != False # == True

оператор больше

True > False # == True

оператор больше или равно

True >= True # == True

оператор меньше

True < True # == False

оператор меньше или равно

True <= True # == True

Оператор not

```
>>> not True  
False
```

```
>>> not 1  
False
```

```
>>> not []  
True
```

Операторы and и or

примеры использования and

```
>>> True and False  
False
```

```
>>> [] and True  
[]
```

```
>>> True or False  
True
```

```
>>> [] or {}  
{}
```

примеры использования or

Ветвление

```
if expression1:  
    do_something1()  
  
elif expression2:  
    do_something2()  
  
elif expression3:  
    do_something3()  
  
else:  
    do_another_thing()
```

Другие варианты ветвления

без elif

```
if condition:  
    do_something()  
  
else:  
    do_another()
```

без else

```
if condition1:  
    do_if_case()  
  
elif condition2:  
    do_elif_case()
```

только if

```
if condition:  
    do_if_case()
```

Замечание 1: bool в условии

Пример, как делать не надо

```
if bool(condition):
```

...

Пример python-way кода

```
if condition:
```

...

Замечание 2: сравнение с bool

не нужно сравнивать значения явно

```
if my_variable == True:
```

...

такая проверка имеет место

однако количество ситуаций, в которых она будет

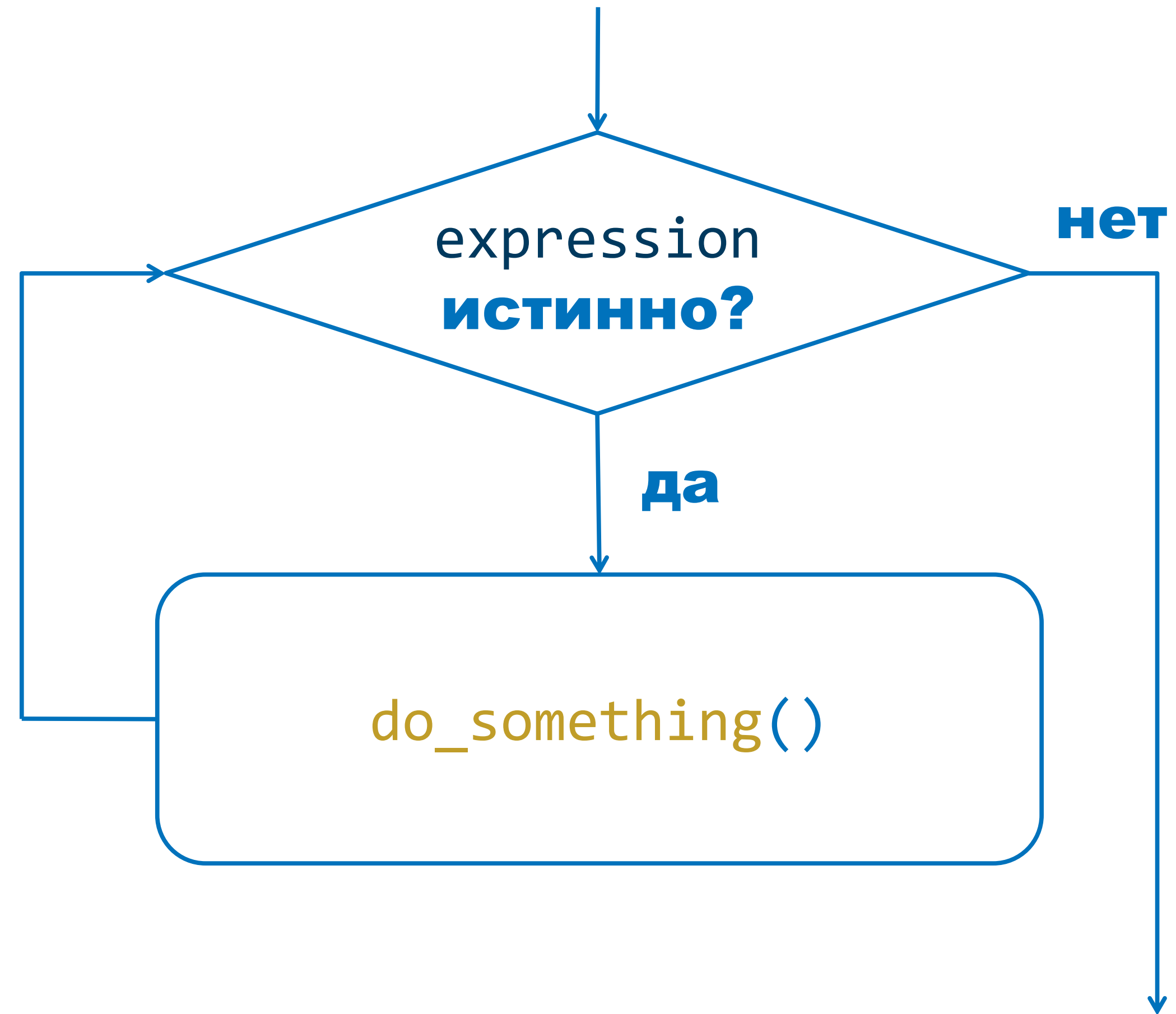
полезна, невелико

```
if my_variable is True:
```

...

Цикл *while*

```
while expression:  
    do_something()
```



continue

без использования continue

```
while condition:
    if get_condition():
        do_something()
        do_another_thing()
        update_condition()
```

```
while condition:
    if not get_condition():
        continue

    do_something()
    do_another_thing()
    update_condition()
```

с использованием continue

break n else

```
while condition:
    if not get_break_condition():
        break

    do_something()
    do_another_thing()
    update_condition()

else:
    finish_while_loop()
```


Практическая часть