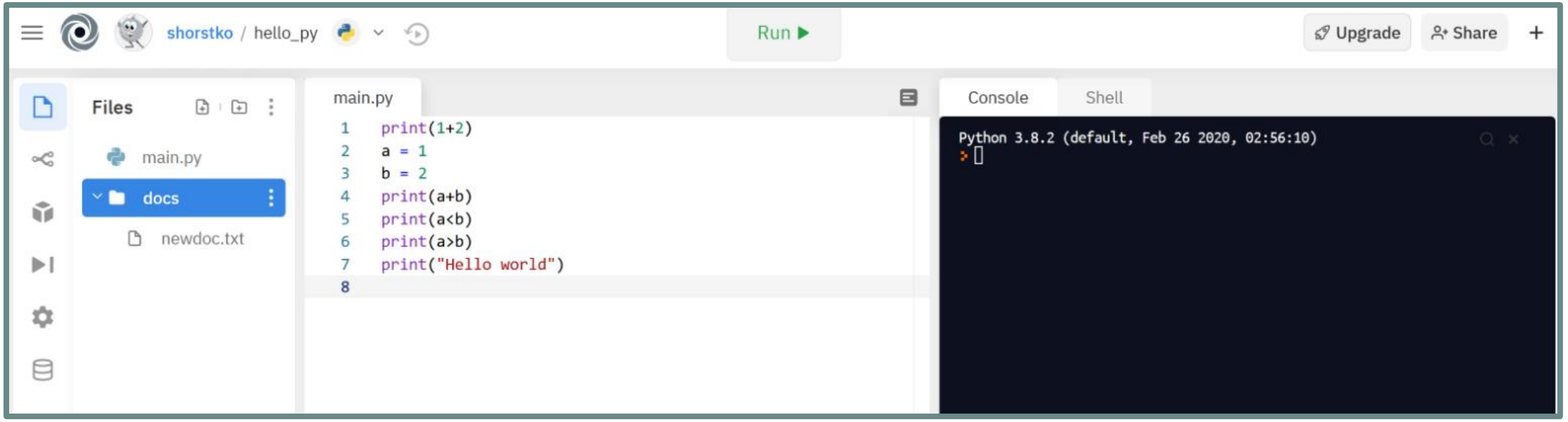
Лекция 1. Типы данных

Простые типы данных: числа, логический (булев) тип, тип None. Изменяемые и неизменяемые типы данных. Преобразование типов. Проверка типа: универсальные методы (type, isinstance), специфичные методы (isalpha, isdigit...). Операторы в Python.

Строки и операции над ними. Вывод на экран (print), ввод с клавиатуры (input). Продвинутый вывод на экран (format, f-строки). Кавычки и апострофы в функции print(). Префиксы строк

Repl.it – онлайн среда

Repl.it – это онлайн-среда программирования, которая позволяет писать и запускать код сразу из браузера. Данные хранятся в облаке. Можно расшарить ссылку и в онлайн-режиме показывать изменения кода



Типы данных в python

По организации

- Простые (числа, строки, булевы)
- Коллекции (кортежи, списки, словари, множества и их производные)
- Произвольные (классы)
- Тип None

По изменямости

- Неизменяемые (числа, строки, булевы, кортежи)
- Изменяемые (списки, словари, множества)

Простые типы

Простые типы данных: числа, логический (булев) тип, тип None.
Изменяемые и неизменяемые типы данных

Числа и операции с ними

a + b	**Сложение** можно складывать int и float между собой	3 + 1 = 4 3 + 1.2 = 4.2
a - b	**Вычитание** можно вычитать int и float между собой	3 - 1 = 2 3 - 1.2 = 1.8 1 – 3 = -2
a * b	**Умножение** можно умножать int и float между собой	3 * 1 = 3 3 * 1.2 = 3.6
a ** b	**Возведение в степень** можно использовать int и float в любых сочетаниях	3 ** 3 = 27 3 ** 1.2 = 3.7371928188465517

Операции с числами

a / b	**Деление** можно делить int и float друг на друга	3 / 1 = 3 3 / 1.2 = 2.5
a // b	**Деление нацело – оставляем целую часть** можно делить int и float друг на друга	3 // 1 = 3 3 // 1.2 = 2.0
a % b	**Остаток от деления – оставляем дробную часть** можно делить int и float друг на друга	3 % 1 = 0 3 // 1.2 = 2.0
a / b	**Деление** можно делить int и float друг на друга	3 / 1 = 3 3 / 1.2 = 2.5
round(x, n)	**Округление** указать количество знаков после точки	round(2.65, 1) round(2.75, 1)

Операции с числами (продолжение)

abs(a)	Получение модуля числа модуль числа – это его значение без знака	abs(3) = 3 abs(-3) = 3 abs(-1.2) = 1.2
-a	Изменение знака числа на противоположный	a = 3, -a ⟶ -3, -a ⟶ 3 a = 1.2, -a ⟶ -1.2, -a ⟶ 1.2
int(a)	Приведение к типу «целое»	a = 1.0 int(a) ⟶ 1
float(a)	Приведение к типу «вещественное)	a = 1 float(a) ⟶ 1.0

Логический тип (bool)

В python пишется как True или False (обязательно с большой буквы).

Преобразования в bool:

В True	Ненулевое число: bool(3) → True, bool(1.2) → True Непустую строку: bool(“hello”) → True Непустой объект, например, список: bool([“мандарины, кг”, 1.3]) → True
В False	Нулевое число: bool(0) → False, bool(0.0) → False Пустую строку: bool(“”) → False Пустой объект, например, список: bool([]) → False Тип None: bool(None) → False

Операции с bool

==	**Двойное равно** проверка, что значения справа и слева одинаковы	3 == 3.0 → False 3 == 3 → True 3 == “3” → False 0 == None → False
!=	**Не равно** проверка, что значения справа и слева разные	3 != 3.0 → True 3 != 3 → False 3 != “3” → True 0 != None → True
is	**Может заменять двойное равно** работает так же, как ==, но используется не в вычислениях	3 is 3.0 → False 3 is 3 → True
is not	**Может заменять «не равно»** проверка, что значения справа и слева разные, но используется не в вычислениях	3 is not 3.0 → True 3 is not 3 → False

Операции с bool (продолжение)

>	**Больше** проверка, что значение слева больше значения справа	3 > 3.0 → False 3 > 3.5 → False “A” > “B” → False “раз” > “два” → True 1.2 > “1”→ ошибка!	int и float одинаковы, это все равно число 3 номер буквы «B» больше номера буквы «A», поэтому условие неверно номер буквы «р», больше номера буквы «д», поэтому условие неверно нельзя сравнить число и строку
<	**Меньше** проверка, что значение слева меньше значения справа	3 < 3.0 → False 3 < 3.5 → True “A” < “B” → True “раз” > “два” → False 1.2 < “1”→ ошибка!	int и float одинаковы, это все равно 3 номер буквы «A» действительно меньше, чем у «B» номер «р» больше «д», поэтому неверно нельзя сравнить число и строку

Операции с bool (продолжение)

>=	Больше или равно проверка, что значение слева больше значения справа либо равно ему	3 >= 3.0 → True 3 >= 3.5 → False	int и float одинаковы, это все равно число 3. в данном случае работает часть условия «равно», а «больше» - не работает
<=	Меньше или равно проверка, что значение слева меньше значения справа либо равно ему	3 <= 3.0 → True 3 <= 3.5 → False	int и float одинаковы, это все равно 3. в данном случае работает часть условия «равно», а «меньше» - не работает

Операции с bool (продолжение)

a + bool	**Сложение** можно складывать int/float и bool между собой	3 + True = 4 3 + False = 3
a - bool	**Вычитание** можно вычитать int/float и bool между собой	3 - True = 2 3 – False = 3
a * bool	**Умножение** можно умножать int/float и bool между собой	3 * True = 3 3 * False = 0
a / bool bool / a	**Деление** внимание: при делении на True результат будет типа float. на False делить нельзя, т.к. это деление на 0	3 / True = 3.0 3 / False = ошибка!

Тип None

None означает «ничего».

Тип **None** нужен, когда мы еще не определились, что будет дальше, но нам уже нужно зарезервировать место под будущее значение.

Эквивалентен null в C++

Строки

Строки и операции над ними

Строки

- Строка – это последовательность символов, к каждому из которых можно обратиться по индексу. Индексы начинаются с 0. Например: “hello”[0] -> “h”
- Строки в python можно упаковать тремя вариантами: “hello” (кавычки), ‘hello’ (апострофы), ’’’hello’’’ (тройной апостроф)
- При помощи апострофов можно сделать и многострочную строку, и многострочный комментарий, и даже писать документацию:

```
’’’А у нас в квартире газ. А у вас?
А у нас – водопровод. Вот!’’’
```

Операции со строками

s1 + s2	**Конкатенация (сложение)** строки можно складывать – «сцеплять» между собой	“Мама” + “ “ + “мыла” ⟶ “Мама мыла”
s[n]	**Доступ по индексу** получить любой символ из строки. Можно идти с начала (n положительное) или с конца (n отрицательное)	“Мама”[0] ⟶ “М” “Мама”[3] ⟶ “а” “Мама”[-1] ⟶ “а” *[3] и [-1] указывают на один и тот же символ*
s[n1,n2]	**«Срез» строки** «вырезать» часть строки. Можно указывать обе границы диапазона (n1 и n2) либо только левую (откуда начать) или только правую (где закончить)	“Мама”[:2]⟶ “Ма” “Мама”[2:] ⟶ “ма” “Мама”[1:3] ⟶ “ам”

Функции строк

len(s)	**Длина строки** подсчитывает количество символов в строке	len(“Мама”) → 4
s.find(str)	**Поиск подстроки в строке** возвращает номер первого вхождения или -1	“Мама”.find(“ма”) → 2
s.replace(шаблон, замена)	**Замена подстроки в строке** шаблон – что искать, замена – на что заменить	“Мама”.replace(“а”, “у”) → “Муму”
s.split(шаблон)	**Разбиение строки** строка разбивается по указанному шаблону. Например, чтобы подсчитать количество слов в предложении, можно разбить его по пробелу	“Мама мыла раму”.split(“ “) → [“Мама”, “мыла”, “раму”]

Функции строк (продолжение)

s.isdigit()	**Состоит ли строка только из цифр?** проверяет, можно ли превратить строку в число	“Мама”.isdigit() → False “2021”.isdigit() → True
s.isalpha()	**Состоит ли строка только из букв?** проверяет, что в строке только буквы	“Мама”.isalpha() → True “Мама2021”.isalpha() → False
s.isalnum()	**Состоит ли строка только из букв и цифр?** проверяет, что в строке только буквы и цифры	“Мама2021”.isalnum() → True
s.capitalize()	Переводит первую букву строки в верхний регистр, а остальные – в нижний регистр пригодится для исправления предложений	“мама мыла раму.” → “Мама мыла раму.”

Функции строк (продолжение)

s.istitle()	Начинаются ли все слова в строке с большой буквы? пригодится для проверки ФИО	“Мама мыла раму”.istitle() → False “Иванов Иван Иванович”.istitle() → True
s.title()	Переводит первую букву каждого слова в верхний регистр, а остальные – в нижний пригодится для исправления ФИО	“Иванов иван иванович”.capitalize() → “Иванов Иван Иванович”
s.islower() s.isupper()	Состоит ли строка только из символов в нижнем регистре / в верхнем регистре? пригодится для проверки аббревиатур	“Мама”.isupper() → False “IT”.isupper() → True
s.upper() s.lower()	Переводит все слова в верхний регистр / нижний регистр	“Мама”.upper() → “MAMA” “Мама”.lower() → “мама”

Функции строк (продолжение)

s.join(список)	Склеивает в одну строку все элементы списка через разделитель s можно составить фразу из отдельных слов. Например, собрать ФИО, если из базы данных вам пришли отдельно имя, фамилия и отчество	“ “.join([“Иванов”, “Иван”, “Иванович”]) → “Иванов Иван Иванович”
s.strip()	«Чистит» строку от пробелов по краям	“ Мама ”.strip() → “Мама”
s.format()	Можно подставить в строку нужные части пригодится, когда вам нужно подставлять данные, которые меняются. Для подстановки в нужные места строки нужно вставить фигурные скобки: {}	“Меня зовут {} и я {}”.format(“Лена”, “программист”) → “Меня зовут Лена и я программист”

Взаимодействие строк с другими типами

int(s) str(i)	Можно преобразовать строку в целое число и целое число в строку	int(“33”) → 33 str(33) → “33” int(“36.6”) → ошибка!
float(s) str(f)	Можно преобразовать строку в вещественное число и вещественное число в строку	float(“36.6”) → 36.6 str(36.6) → “36.6” float(“33”) → 33.0
bool(s)	Можно проверить, соответствует строка True (непустая строка) или False (пустая строка)	bool(“Мама”) → True bool(“”) → False
s1 * N	Дублирование («умножение») строку можно «умножить» на число. В результате она повторится N раз	“Ла” * 3 → “ЛаЛаЛа”

Спецсимволы в строках

\n	Перевод строки после этого символа начинается новая строка	“Мама мыла раму.\nПросто супермама!” → Мама мыла раму. Просто супермама!
\t	Горизонтальная табуляция делает отступ по горизонтали	“Мама мыла раму.\tПросто супермама!” → Мама мыла раму. Просто супермама!
\v	Вертикальная табуляция делает отступ по вертикали Предупреждение: именно эта табуляция может стоять только в начале или конце строки!	“Мама мыла раму.” "\vПросто супермама!” → Мама мыла раму. Просто супермама!
\	Символ экранирования показывает, что за ним следует команда	Если мы хотим напечатать сам символ «\», нужно написать его два раза: “\\”

Спецсимволы в строках (продолжение)

Если нужен символ кавычки – пишем строку в апострофах	‘Посмотри сериал “Программисты” – очень круто!’ → Посмотри сериал “Программисты” – очень круто!	Если нужен символ кавычки – пишем строку в апострофах
Если нужен символ апострофа – пишем строку в кавычках	“Функция print выводит строки в апострофах: ‘” → Функция print выводит строки в апострофах: ‘	Если нужен символ апострофа – пишем строку в кавычках
Используем тройной апостроф	‘’’Редакторы так любят комбинировать кавычки… например, “ООО ‘Прогресс’”…’’’ Редакторы так любят комбинировать кавычки… например, “ООО ‘Прогресс’”…	Используем тройной апостроф
Используем символ экранирования	“Посмотри сериал \“Программисты\” – очень круто!” → Посмотри сериал “Программисты” – очень круто!	Используем символ экранирования

Проверка типа

Проверка типа: универсальные методы (type, isinstance), специфичные методы (isalpha, isdigit...)

Как узнать тип?

Тип данных нам подскажет функция **type()**. А узнать, относится ли наше значение или переменная к определенному типу – **isinstance()**

type(3)	<class ‘int’>
type(3.0)	<class ‘float’>
type(True) type(False)	<class ‘bool’>
type(None)	<class ‘NoneType’>

isinstance(3, int)	True
isinstance(3.0, (int, float))	True
num = 3 isinstance(num, int)	True
num = 3.0 isinstance(num, int)	False

Проверка типа

Помимо **isinstance()** есть функции проверки конкретных типов

	str.isdecimal()	**str.isdigit()**
“12345”	True	True
“①²³4₅”	False	True
	возвращает True, если все символы в строке являются цифрами 0..9	возвращает True, если все символы в строке являются цифрами 0..9, а также цифрами в виде надстрочных индексов

Проверка типа конверсией

Есть функции явного преобразования типов. **Осторожно с числами!**
Если на входе неподходящий тип, функция выкинет исключение!

	int(str)	**float(str)**
“12345”	12345	12345.0
“12345.6”	ошибка ValueError	12345.6
“1²345”	ошибка ValueError	ошибка ValueError
	int(str) – преобразование в целое число	преобразование в вещественное число

Операторы

Виды операторов в python с примерами

Операторы в python

Когда нам необходимо сделать что-либо с двумя и более значениями, эти значения называются **операндами**, а графическое обозначение операции, которую производят над этими значениями, называется **оператором**.

Слабоумие & Отвага

Виды операторов в python

- **Арифметические операторы**. Позволяют производить расчеты над числами
- **Операторы сравнения**. Сравнение двух или более значений с использованием арифметики
- **Операторы присваивания**. Присваивают переменным значения (в том числе значение как результат вычислений с использованием других операторов)
- **Побитовые операторы**. Работают с двоичным представлением данных в виде нулей и единиц (битов). В этом курсе вы с ними не встретитесь
- **Логические операторы**. Логическое сравнение значений («И», «ИЛИ», «НЕ»)
- **Операторы членства**. Устанавливают принадлежность элемента последовательности
- **Операторы тождественности**. Логическое сравнение операндов (проверка совпадения адреса в памяти, а не значения)

Арифметические операторы

+	Сложение	**3 + 1.2 = 4.2**
-	Вычитание	3 - 1.2 = 1.8
*	Умножение	3 * 1.2 = 3.6
**	Возведение в степень	3 ** 3 = 27 3 ** 1.2 = 3.7371928188465517
/	Деление (на ноль делить нельзя)	3 / 1.2 = 2.5
//	Деление нацело – оставляем только целую часть	3 // 1 = 3 3 // 1.2 = 2.0
%	Остаток от деления – оставляем только дробную часть	3 % 1 = 0 3 % 1.2 = 0.5

Операторы сравнения

==	Проверяет, что оба операнда равны	3 == 3 → True True == False → False
!=	Проверяет, что оба операнда **не равны**	1 != 2 → True “Мама” != “мама” → True
>	Проверяет, что значение левого операнда **больше**, чем правого	3 > 2 → True “А” > “Б” → False
<	Проверяет, что значение левого операнда **меньше**, чем правого	3 < 2 → False “А” < “Б” → True
>=	Проверяет, что значение левого операнда **больше либо равно** значению правого операнда	3 >= 3.0 → True
<=	Проверяет, что значение левого операнда **меньше либо равно** значению правого операнда	3 <= 3.0 → True

Операторы присваивания

=	Присваивает значение правого операнда левому	my_name = “Лена”
+=	Складывает значения левого и правого операнда и присваивает результат левому операнду	a = 1 b = 2 a += b или a = a + b ⟶ 3
-=	Вычитает значение правого операнда из левого операнда и присваивает результат левому операнду. Выполняется как для чисел, так и для множеств	a = 2 b = 1 a -= b или a = a - b ⟶ 1
*=	Умножает левый операнд на правый и присваивает результат левому операнду	a = 1 b = 2 a *= b или a = a * b ⟶ 2
**=	Возводит левый операнд в степень правого и присваивает результат левому операнду	a = 1 b = 2 a **= b или a = a ** b ⟶ 1

Операторы присваивания (продолжение)

/=	Делит левый операнд на правый и присваивает результат левому операнду	a = 1 b = 2 a /= b или a = a / b → 0.5
%=	Получает остаток от деления (все, что не делится нацело) левого операнда на правый и присваивает результат левому операнду	a = 1 b = 2 a %= b или a = a % b → 1
//=	Получает целую часть от деления левого операнда на правый и присваивает результат левому операнду	a = 1 b = 2 a //= b или a = a // b → 0
|=	Вычисляет объединение множеств в правом операнде и в левом операнде и присваивает результат левому операнду	set1= {1, 2} set1 |= {4, 5} | {3} → set1 = {1,2,3,4,5}
&=	Вычисляет пересечение множеств в правом операнде и в левом операнде и присваивает результат левому операнду	set1= {1, 2} set1 &= ({1,5}, {1,3} → set1 = {1}
^=	Вычисляет симметрическую разность множеств в правом операнде и в левом операнде и присваивает результат левому операнду	set1= {1, 2} set1 ^= {3, 2} → set1 = {1,3}

Логические операторы

and	Логический оператор "И". Условие будет истинным, только если оба операнда True или не False и не 0 (операнд со значением False как бы «умножает» второй операнд на 0)	True and True → True True and “a” → True True and False → False False and False → False True and 0 → False
or	Логический оператор "ИЛИ". Условие будет истинным, если хотя бы один из операндов имеет значение True (или не False и не 0)	True or True → True True or “a” → True True or False → True False or False → False True or 0 → True
not	Логический оператор "НЕ". Изменяет логическое значение операнда на противоположное	not True → False not False → True not 0 → True not 1 → False not “a” → False

Операторы членства

in	Возвращает True, если элемент присутствует в последовательности, иначе возвращает False	“ма” in “Мама” ⟶ True 1 in [1, 2, 3] ⟶ True “раз” in [“раз”, “два”] ⟶ True ”три” in [“раз”, “два”] ⟶ False
not in	Возвращает True, если элемента нет в последовательности. То есть результаты противоположны результатам оператора in	“ма” not in “Мама” ⟶ False “ра” not in “Мама” ⟶ True ”три” not in [“раз”, “два”] ⟶ True

Операторы тождественности

is	Возвращает True, если оба операнда указывают на один объект (на один и тот же адрес в памяти*)	l1 = [1, 2] l2 = l1 l1 is l2 → True или id(l1) == id(l2) l2 = l1.copy() l1 is l2 → False или id(l1) != id(l2)
is not	Возвращает True, если операнды указывают на разные объекты (разные адреса в памяти)	l1 = [1, 2] l2 = l1 l1 is not l2 → False l2 = l1.copy() l1 is not l2 → True

* адрес объекта (переменной) в памяти можно получить функцией **id():**
id(l1) -> 140472391630016 (уникальное число)
Функцию id() мы будем изучать, когда дойдем до классов

Изменяемые и неизменяемые типы

Изменяемые и неизменяемые типы

Неизменяемые типы данных (immutable) – это числа (int, float и пр.), строки (str), логический тип (bool), кортежи (tuple), статичные множества (frozenset).

Изменяемые типы данных (mutable) – это список (list), словарь (dict), множество (set) и многие другие «дополнительные» типы, которые не встроены в Python, но вы можете их добавить, если они вам нужны (вспомните урок 3).

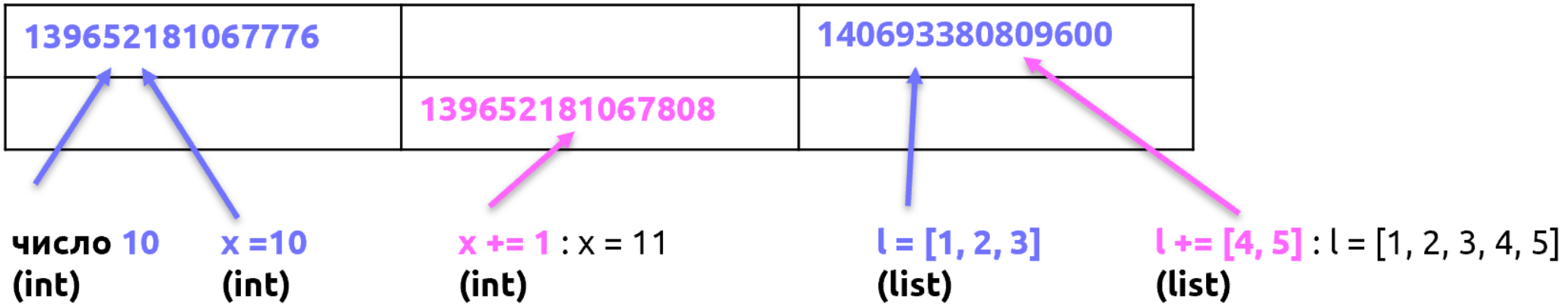
Разница в том, что при попытке изменить **неизменяемый тип** данных он **создается заново** и его адрес в памяти меняется. А вот **изменяемые типы** данных **сохраняют свой адрес постоянным**.

Изменяемые и неизменяемые типы (продолжение)

Любой тип данных в Python является **объектом**: помимо значения (например, **«Мама»** для строки или **4** для числа), у него есть свои свойства и методы.

Чтобы сохранить такой сложный объект в памяти и потом его найти, объекту, как и человеку, необходим **адрес** его «места жительства».

В Python адрес объекта – это очень длинное число: **139652181067776**. Узнать адрес можно через **id()**.



Ввод с клавиатуры. Вывод на экран

Вывод на экран (print), ввод с клавиатуры (input). Продвинутый вывод на экран (format, f-строки). Кавычки и апострофы в функции print(). Префиксы строк. Создание превью текста

Ввод с клавиатуры: input()

- **input()** всегда возвращает строку. Даже если вы вводили число, для программы это все равно набор символов
- если пользователь вводил целое число, используйте **int()** для преобразования
- если пользователь вводил вещественное число (с плавающей точкой), используйте **float()** для преобразования
- всегда подсказывайте пользователю, что от него требуется, при помощи параметра в функции, например: **input("Введите свое имя и возраст через пробел")**
- **input()** должна что-то вернуть, поэтому создавайте переменную, в которую присвоите результат, например: **my_data = input("Введите свое имя и возраст через пробел")**
- помните, что строки чувствительны к регистру и, например, "Да" и "да" – это разные строки. Если вам нужен ответ от пользователя, лучше дополнительно привести результат к нижнему регистру: **my_data = input("Введите свое имя и возраст через пробел").lower()**

Вывод на экран: print()

- Функция **print()** выводит информацию на экран. Сделать это можно несколькими способами:
- В качестве параметра перечислить через запятую все данные, которые вы хотите вывести: **print(“Мой бюджет получился”, my_budget, “руб”).** Данные могут быть разных типов.
- Подстановка значений. Вы пишете «заготовку» строки и в нужных местах ставите **{}**. Вызываете для этой строки функцию **format()** и внутри нее, через запятую, пишете подряд значения, которые хотите подставить:
  print(“Мой бюджет получился {} руб”.format(my_budget)). Значения подставятся в том порядке, в котором вы их написали. Если хотите задать порядок «вручную», укажите число внутри **{}**, например:
  print(“Мой бюджет получился {0} руб. Это на {1} больше минимального”.format(my_budget, my_budget - budget)).
- f-строки. Это более компактная вариация пункта 2. Вы можете подставлять ваши значения из переменных сразу внутри «заготовки» строки. Для этого перед строкой добавляйте букву f:
  print(f“Мой бюджет получился {my_budget} руб”).

Кавычки и апострофы в print()

- Внутри функции **print()** действуют те же правила использования кавычек и апострофов, что и для строк.
- Если в строке внутри **print()** встречаются кавычки “ – запишите саму строку в апострофы ‘. Например: **print(‘Мамин рецепт “шарлотки” самый лучший!’)**.
- Если в строке встречаются апострофы (например, в английском языке), запишите строку в кавычках: **print(“Mom’s apple pie is the best!”)**.

Префиксы строк

Префикс – это дополнительная специальная буква, которая определяет, как будет себя вести строка. Вы можете использовать:

- **f** (форматированная строка, f-строка), в которую можно подставлять значения **print(f“Мой бюджет получился {my_budget} руб”)**
- **r** (raw, «сырые» строки). С таким префиксом Python читает строку в точности так, как она написана, не различая символов экранирования. «Сырые» строки, например, очень удобны для записи путей к файлам, в которых в качестве разделителя имен папок уже используется символ экранирования “\”: **my_dir = r”c:\homework1”**. Иначе пришлось бы написать **my_dir = ”c:\\homework1”**

Префиксы строк (продолжение)

- **fr** («сырые» форматированные строки). Сочетают преимущества «сырых» строк (как вижу, так и пишу) и возможность подстановки значений. Опять же, удобно для работы с путями на диске: `my_dir = fr”c:\homework{number}”`.

  Вместо **number** можно подставлять номер текущего домашнего задания, а префикс **r** избавит от необходимости писать символ “\” дважды.

Префиксы строк (продолжение)

Существуют и более специальные префиксы:

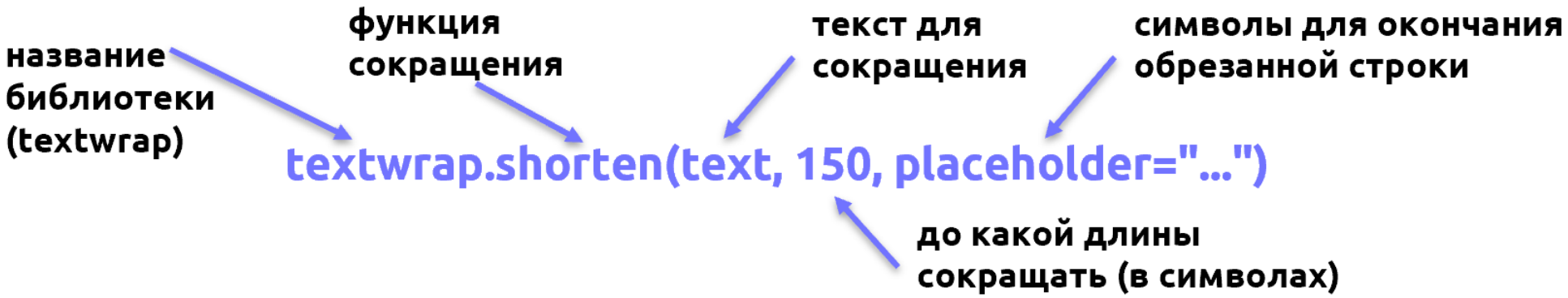
- **b** (байтовые строки). Такие строки записываются не символами, а их числовыми кодами. Например, слово **«Мама»** будет выглядеть вот так: **b”\xcc\xe0\xec\xe0”**. Такие строки нужны в основном для сохранения данных на диск, так как некоторые символы слишком специфичны и зависят от конкретного языка строки (например, русский, немецкий, арабский, …)
- **u** (Unicode-строки). В основном нужны для совместимости между python 2 и 3 или решения проблем вывода в консоль. Просто допишите символ **“u”** перед вашей строкой: **print(u”Мама”)**

Создание превью текста

Превью – это короткий фрагмент текста, по которому можно понять, о чем идет речь. В python можно сделать превью при помощи библиотеки **textwrap** и функции **textwrap.shorten()**.

Первым значением в функцию передается строка, которую вы хотите сократить, вторым – количество символов, до которой ее обрежут, и третьим – символы для окончания обрезанной строки, например, традиционное многоточие.

Для использования textwrap обязательно добавьте в начало кода строку для подключения новой библиотеки для Python: import textwrap



Создание превью текста (продолжение)

```
4   import textwrap
5
6   long_text = '''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 набор полезных функций. Википедия'''
7   print(textwrap.shorten(long_text, 150, placeholder="..."))
```

```
Python — высокоуровневый язык программирования общего назначения, ориентированный на
повышение производительности разработчика и читаемости кода....
```

Домашнее задание

текст дз