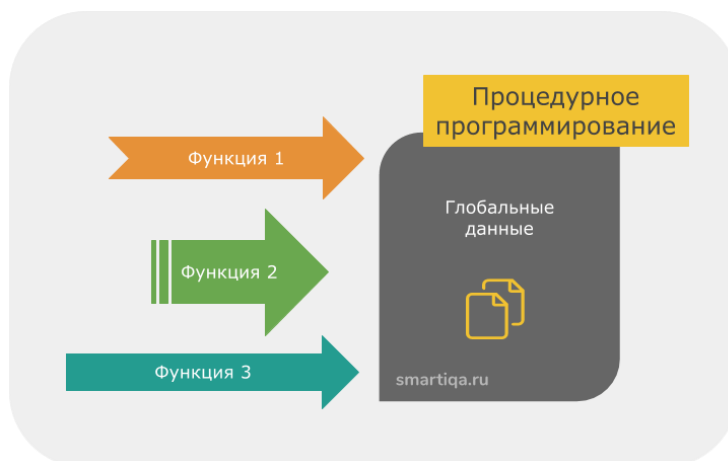


Лабораторная работа №10: «ООП»

Теоретическое введение

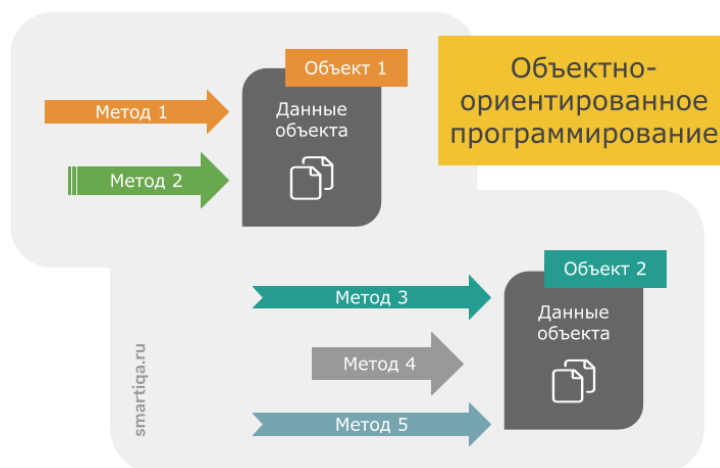
Процедурное программирование – это написание функций и их последовательный вызов в некоторой главной(main) функции.

Главным является код для обработки данных.



Объектно-ориентированное программирование (ООП) – основано на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определённого класса, а классы образуют иерархию наследования.

Главное в программе – это данные. Именно они определяют, какие методы будут использоваться для их обработки.



10.1. Понятие класса

Класс = Свойства + Методы

- **Класс** описывает множество объектов, имеющих общую структуру и обладающих одинаковым поведением. Класс – это шаблон кода, по которому создаются объекты. Т.е. сам

по себе класс ничего не делает, но с его помощью можно создать объект и уже его использовать в работе.

Данные внутри класса делятся на свойства и методы.

- **Свойства класса**(они же поля) – это характеристики объекта класса.
- **Методы класса** – это функции, с помощью которых можно оперировать данными класса.
- **Объект** – это конкретный представитель класса. Объект класса **экземпляр класса** – это одно и то же.

Класс определяется с помощью ключевого слова **class**:

```
1 class название_класса:
2     атрибуты_класса
3     методы_класса
```

Пример простейшего класса:

```
1 class Person:
2     pass
```

В данном случае в классе не определяется никаких методов или атрибутов. Однако поскольку в нем должно быть что-то определено, то в качестве заменителя функционала класса применяется оператор **pass**.

После создания класса можно определить объекты этого класса.

```
1 class Person:
2     pass
3
4 tom = Person()      # определение объекта tom
5 bob = Person()      # определение объекта bob
```

Person() – вызов конструктора, который возвращает объект класса Person

10.2. Методы классов

Методы класса фактически представляют функции, которые определены внутри класса и которые определяют его поведение.

```
1 class Person:      # определение класса Person
2     def say_hello(self):
3         print("Hello")
4
5 tom = Person()
6 tom.say_hello()    # Hello
```

При определении методов любого класса следует учитывать, что все они должны принимать в качестве первого параметра ссылку на текущий объект, который согласно условиям называется **self**.

Через эту ссылку внутри класса мы можем обратиться к функциональности текущего объекта. Но при самом вызове метода этот параметр не учитывается.

Если метод должен принимать другие параметры, то они определяются после параметра **self**, и при вызове подобного метода для них необходимо передать значения:

```
1 class Person: # определение класса Person
2     def say(self, message): # метод
3         print(message)
4
5
6 tom = Person()
7 tom.say("Hello, Im Tom")
```

Через ключевое слово **self** можно обращаться внутри класса к функциональности текущего объекта:

```
1 self.атрибут    # обращение к атрибуту
2 self.метод      # обращение к методу
```

```
1 class Person:
2
3     def say(self, message):
4         print(message)
5
6     def say_hello(self):
7         self.say("Hello work") # обращаемся к выше определенному методу say
8
9
10 tom = Person()
11 tom.say_hello() # Hello work
```

Здесь в одном методе - `say_hello()` вызывается другой метод - `say()`:

10.3. Конструкторы

Для создания объекта класса используется конструктор. Так, выше когда мы создавали объекты класса `Person`, мы использовали конструктор по умолчанию, который не принимает параметров и который неявно имеют все классы.

Однако мы можем явным образом определить в классах конструктор с помощью специального метода, который называется `__init__()` (по два прочерка с каждой стороны).

```
1 class Person:
2     # конструктор
3     def __init__(self):
4         print("Создание объекта Person")
5
6     1 usage
7     def say_hello(self):
8         print("Hello")
9
10 tom = Person() # Создание объекта Person
11 tom.say_hello() # Hello
```

В качестве первого параметра конструктор, как и методы, также принимает ссылку на текущий объект – `self`. Обычно конструкторы применяются для определения действий, которые должны производиться при создании объекта.

10.4. Атрибуты объекта

Атрибуты хранят состояние объекта. Для определения и установки атрибутов внутри класса можно применять словос `self`.

```
1 class Person:
2
3     def __init__(self, name):
4         self.name = name # имя человека
5         self.age = 1     # возраст человека
6
7
8 tom = Person("Tom")
9
10 # обращение к атрибутам
11 # получение значений
12 print(tom.name) # Tom
13 print(tom.age)  # 1
14 # изменение значения
15 tom.age = 37
16 print(tom.age)  # 37
```

Можно определять атрибуты вне класса – Python позволяет сделать это динамически вне кода:

```

1 class Person:
2
3     def __init__(self, name):
4         self.name = name      # имя человека
5         self.age = 1          # возраст человека
6
7
8 tom = Person("Tom")
9
10 tom.company = "Microsoft"
11 print(tom.company) # Microsoft

```

В то же время подобное определение чревато ошибками. Например, если мы попытаемся обратиться к атрибуту до его определения, то программа сгенерирует ошибку.

Для обращения к атрибутам объекта внутри класса в его методах также применяется слово `self`:

```

1 class Person:
2
3     def __init__(self, name):
4         self.name = name      # имя человека
5         self.age = 1          # возраст человека
6
7     def display_info(self):
8         print(f"Name: {self.name} Age: {self.age}")
9
10
11 tom = Person("Tom")
12 tom.display_info()           # Name: Tom Age: 1

```

Создание нескольких объектов:

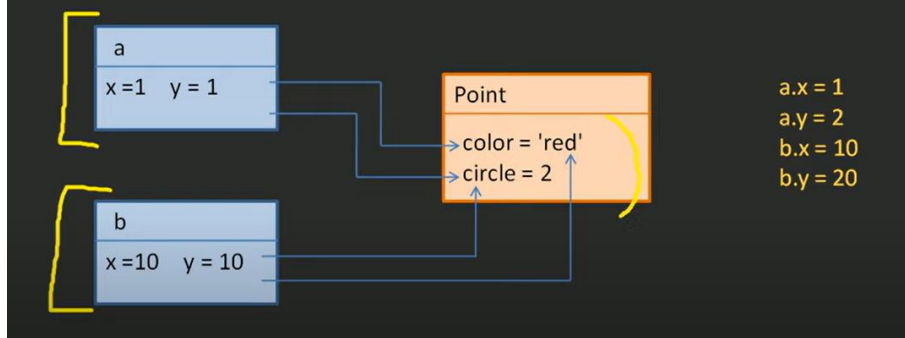
```

1 class Person:
2
3     def __init__(self, name):
4         self.name = name      # имя человека
5         self.age = 1          # возраст человека
6
7     def display_info(self):
8         print(f"Name: {self.name} Age: {self.age}")
9
10
11 tom = Person("Tom")
12 tom.age = 37
13 tom.display_info()           # Name: Tom Age: 37
14
15 bob = Person("Bob")
16 bob.age = 41
17 bob.display_info()           # Name: Bob Age: 41

```

Пример: рассмотрим класс «Точка» и его экземпляры:

Атрибуты классов и объектов



```
3.py x
pythonProject5 1 class Point:
                  2     "класс Точка"
                  3     color = "red"
                  4     circle = 2
                  5     print(Point.__dict__)# |
                  6

Run 3 x

C:\Users\ludag\PycharmProjects\pythonProject5\Scripts\python.exe C:\l
{'__module__': '__main__', '__doc__': 'класс Точка', 'color': 'red',
```

- `__doc__` – содержит строку с описанием класса;
- `__dict__` – содержит набор атрибутов экземпляра класса.

Немного работы в консоли:

```
3.py x
pythonProject5 1 class Point:
                  2     "класс Точка"
                  3     color = "red"
                  4     circle = 2
                  5     print(Point.__dict__)#

Python Console x
>>> class Point:
...     color = "red"
...     circle = 2
...
>>> Point.color="black"
>>> a= Point()
>>> d=Point()
>>> Point.circle
2
>>> a.color="red"
>>> Point.x = 10
>>> d=b
>>> |

a = {Point} <__main__.Point object at 0x000002801E35EE80>
  11 circle = {int} 2
  11 color = {str} 'red'
  11 x = {int} 10
  Protected Attributes
d = {Point} <__main__.Point object at 0x000002801E35E8B0>
  11 circle = {int} 2
  11 color = {str} 'black'
  11 x = {int} 10
  Protected Attributes
Special Variables
  __builtins__ = {dict: 156} {'ArithmeticError': <class 'ArithmeticError'>, 'Asser
  11 __doc__ = {NoneType} None
  11 __file__ = {str} '<input>'
  11 __loader__ = {SourceFileLoader} <_frozen_importlib_external.SourceFileLoa
  11 __name__ = {str} '__main__'
  11 __package__ = {str} ''
  11 __spec__ = {NoneType} None
  Point = {type} <class '__main__.Point'>
  11 circle = {int} 2
```

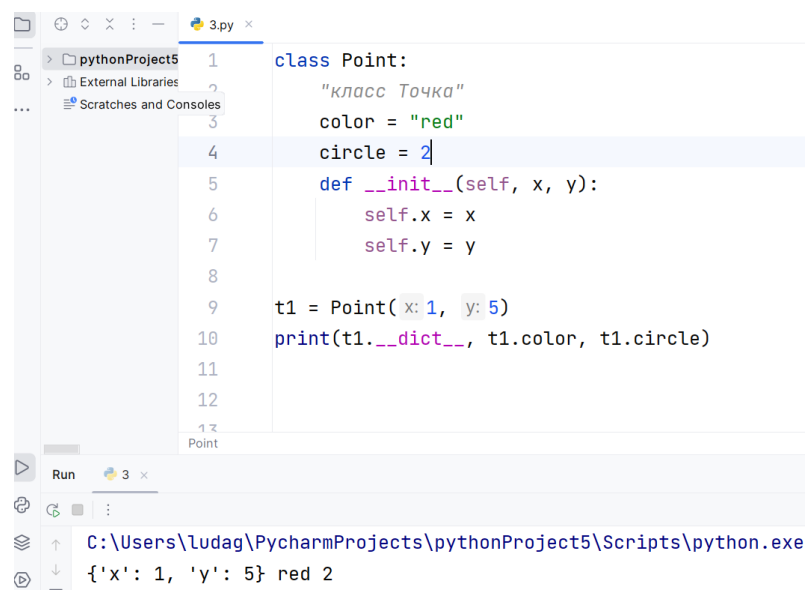
10.5. Магические методы

Магические методы – это методы Python, которые определяют, как ведут себя объекты Python при выполнении над ними обычных операций. Эти методы четко определены с помощью двойного подчеркивания до и после имени метода.

Как правило, эти методы не предназначены для вызова непосредственно в вашем коде; скорее, они будут вызываться интерпретатором во время выполнения программы.

Рассмотрим некоторые.

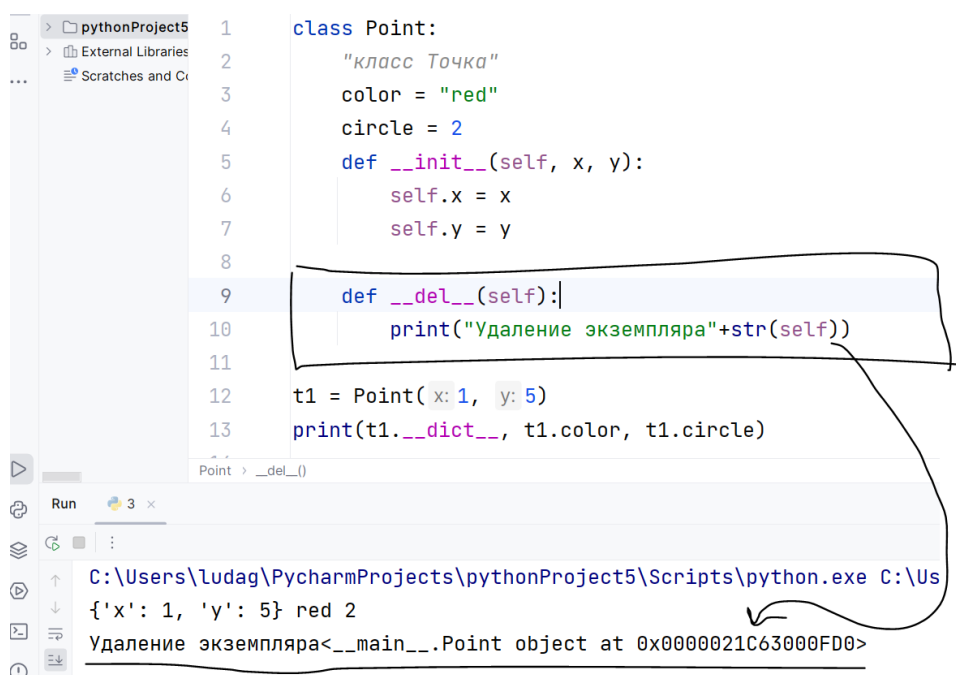
```
__ имя магического метода __  
  
__init__(self) – инициализатор объекта класса  
__del__(self) – финализатор класса
```



The screenshot shows the PyCharm IDE with a file named 3.py. The code defines a class Point with attributes color and circle, and a method __init__. An instance t1 is created with x=1 and y=5. The output of the program is shown in the Run console.

```
1 class Point:  
2     "класс Точка"  
3     color = "red"  
4     circle = 2  
5     def __init__(self, x, y):  
6         self.x = x  
7         self.y = y  
8  
9     t1 = Point(x=1, y=5)  
10    print(t1.__dict__, t1.color, t1.circle)  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

```
Run 3 x  
C:\Users\ludag\PycharmProjects\pythonProject5\Scripts\python.exe  
{'x': 1, 'y': 5} red 2
```



The screenshot shows the same PyCharm IDE with the 3.py file. A new method __del__ has been added to the Point class. The output of the program is shown in the Run console.

```
1 class Point:  
2     "класс Точка"  
3     color = "red"  
4     circle = 2  
5     def __init__(self, x, y):  
6         self.x = x  
7         self.y = y  
8  
9     def __del__(self):  
10        print("Удаление экземпляра"+str(self))  
11  
12    t1 = Point(x=1, y=5)  
13    print(t1.__dict__, t1.color, t1.circle)  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000
```

```
Run 3 x  
C:\Users\ludag\PycharmProjects\pythonProject5\Scripts\python.exe C:\Us  
{'x': 1, 'y': 5} red 2  
Удаление экземпляра<__main__.Point object at 0x0000021C6300FD0>
```

Метод `__call__()` позволяет производить математические операции внутри класса.

```
1 usage
2
3 class FirstClass:
4     def __call__(self, c, d):
5         return c*d
6
7 m = FirstClass()
8 # вывод данных
9
10 print('Первая итерация:', m.__call__(c: 6, d: 12))
11 print('Вторая итерация:', m.__call__(c: 17, d: 11))
12
13
14 Первая итерация: 72
15 Вторая итерация: 187
16
17 Process finished with exit code 0
```

Пример:

Создадим класс «Счетчик»:

```
Project pythonProject5
External Libraries
Scratches and Console

call_py
1 class Counter:
2     def __init__(self):
3         self.i = 0
4
5 c=Counter()
6 print(c.i)
7

К классам можно обращаться
как к функциям, к
экземплярам, если не
прописан метод __call__ - нет.

Run call_
C:\Users\ludag\PycharmProjects\pythonProject5\Scripts\pytho
0
Process finished with exit code 0
```



```
call_py x
1 class Counter:
2     def __init__(self):
3         self.i = 0
4
5     def __call__(self, *args, **kwargs):
6         print("__call__")
7         self.i += 1
8         return self.i
9
10    c=Counter() #создание экземпляра
11    print(c.i) #вывод начального значения
12    c()
13    c()
14    rez = c()
15    print(rez) #вывод измененного значения счетчика
16
```

Благодаря магическому методу `__call__` экземпляры можно вызывать подобно функциям

Классы, ведущие себя таким образом, называют **функторами**.

```
call_py x
1 class Counter:
2     def __init__(self):
3         self.i = 0
4
5     def __call__(self, step = 1, *args, **kwargs):
6         print("__call__")
7         self.i += step
8         return self.i
9
10    c=Counter() #создание экземпляра
11    print(c.i) #вывод начального значения
12    c(2)
13    c(10)
14    rez = c(-5)
15    print(rez) #вывод измененного значения счетчика
16
```

Reader Mode

Counter → `__call__()`

10.6. Порядок выполнения методов

Представим такую ситуацию, что методы в классе должны выполняться последовательно, друг за другом, но из вне, пользователь никак не должен иметь к ним доступа.

Допустим, мы решили сварить пельмени! Какие этапы нас ожидают:

1. начать
2. вскипятить воду
3. положить пельмени в воду
4. ждать 10 мин
5. закончить

Для этого мы объявим класс `cook_pelmeni`, в котором опишем конструктор класса и все остальные методы.

```
inkaps.py
1 class cook_pelmeni:
2
3     def __init__(self):
4         print("начали готовку...")
5
6     def water_ready(self):
7         print("вода вскипела")
8
9     def pelmeni_in_wather(self):
10        print("пельмени в воде")
11
12    def wait(self):
13        print("ждем 10 мин")
14
15    def pelmeni_ready(self):
16        print("пельмени готовы!")
```

Теперь, в конструкторе класса, необходимо вызвать каждый метод в определенном порядке, так как мы не можем взять и поменять нашу последовательность действий местами.

Вызов методов в определённом порядке

```
inkaps.py
class cook_pelmeni:
    def __init__(self):
        print("начали готовку...")
        self.water_ready()
        self.pelmeni_in_wather()
        self.wait()
        self.pelmeni_ready()
    def water_ready(self):
        print("вода вскипела")
```

А теперь мы попробуем вызвать методы вне класса и посмотрим, чем это грозит для нашей программы.

Вызов методов в произвольном порядке

```
17
18     def pelmeni_ready(self):
19         print("пельмени готовы!")
20
21 my_pelmeni=cook_pelmeni()
22
23
24 my_pelmeni.wait()
25 my_pelmeni.pelmeni_ready()
26 my_pelmeni.pelmeni_in_wather()
27 my_pelmeni.water_ready()
```

Run

ждем 10 мин
пельмени готовы!
пельмени в воде
вода вскипела

Process finished with exit code 0

Изменили имена методов

Действия сработали

```

1 class cook_pelmeni:
2     def __init__(self):
3         print("начали готовку...")
4         self.__water_ready()
5         self.__pelmeni_in_wather()
6         self.__wait()
7         self.__pelmeni_ready()
8
9     def __water_ready(self):
10        print("вода вскипела")
11
12    def __pelmeni_in_wather(self):
13        print("пельмени в воде")
14
15    def __wait(self):
16        print("ждем 10 мин")
17
18    def __pelmeni_ready(self):
19        print("пельмени готовы!")
20
21 my_pelmeni=cook_pelmeni()
22
23 my_pelmeni.__wait()
24 my_pelmeni.__pelmeni_ready()
25 my_pelmeni.__pelmeni_in_wather()
26 my_pelmeni.__water_ready()
  
```

Методы теперь закрыты, вызов из вне, недоступен

Run

```

C:\Users\Master\PycharmProjects\pytl
начали готовку...
вода вскипела
пельмени в воде
ждем 10 мин
пельмени готовы!
Traceback (most recent call last):
  File "C:\Users\Master\PycharmProj
    my_pelmeni.__wait()
AttributeError: 'cook_pelmeni' objec
Process finished with exit code 1
  
```

10.7. Принципы ООП: абстракция, инкапсуляция, наследование, полиморфизм

Принцип 1. Абстракция

Абстракция – принцип ООП, согласно которому объект характеризуется свойствами, которые отличают его от всех остальных объектов и при этом четко определяют его концептуальные границы.

Т. е. абстракция позволяет:

- ✓ выделить главные и наиболее значимые свойства предмета.
- ✓ отбросить второстепенные характеристики.



Принцип 2. Инкапсуляция

Абстракция утверждает следующее: «Объект может быть рассмотрен с общей точки зрения».

А инкапсуляция от себя добавляет: «И это единственная точка зрения, с которой вы вообще можете рассмотреть этот объект».

А если вы внимательно посмотрите на название, то увидите в нем слово «капсула». В этой самой «капсуле» спрятаны данные, которые мы хотим защитить от изменений извне.

Инкапсуляция – принцип ООП, согласно которому сложность реализации программного компонента должна быть спрятана за его интерфейсом.

На что обратить внимание?

Отсутствует доступ к внутреннему устройству программного компонента.

Взаимодействие компонента с внешним миром осуществляется посредством интерфейса, который включает публичные методы и поля.



Как в данном случае будет работать инкапсуляция? Она будет позволять нам смотреть на дом, но при этом не даст подойти слишком близко.

Например, мы будем знать, что в доме есть дверь, что она коричневого цвета, что она открыта или закрыта. Но каким способом и из какого материала она сделана, инкапсуляция нам узнать не позволит.

Для чего нужна инкапсуляция?

Инкапсуляция упрощает процесс разработки, т. к. позволяет не вникать в тонкости реализации того или иного объекта.

Повышается надежность программ за счет того, что при внесении изменений в один из компонентов, остальные части программы остаются неизменными.

Становится более легким обмен компонентами между программами.

Инкапсуляция, атрибуты и свойства:

По умолчанию атрибуты в классах являются общедоступными, а это значит, что из любого места программы мы можем получить атрибут объекта и изменить его.

Рассмотрим случай, когда присваивается некорректное значение для атрибута.

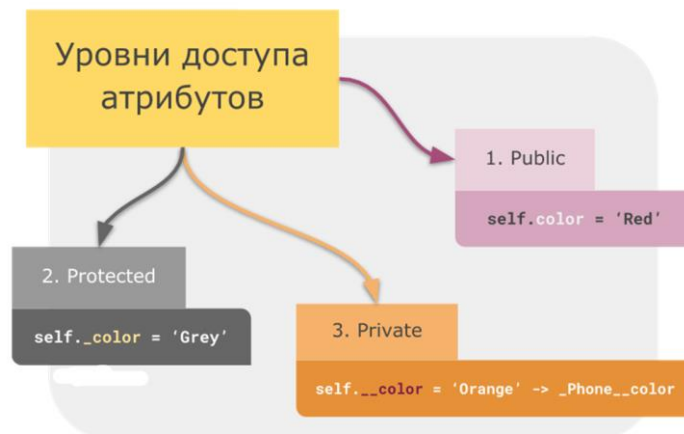
```

1 usage
1 class Person:
2     def __init__(self, name):
3         self.name = name # устанавливаем имя
4         self.age = 1 # устанавливаем возраст
5
6     def display_info(self):
7         print(f"Имя: {self.name}\tВозраст: {self.age}")
8
9
10 tom = Person("Tom")
11 tom.name = "Человек-паук" # изменяем атрибут name
12 tom.age = -129 # изменяем атрибут age
13 tom.display_info() # Имя: Человек-паук    Возраст: -129

```

Подобное поведение нежелательно, поэтому встает вопрос о контроле за доступом к атрибутам объекта.

- **Инкапсуляция** предотвращает прямой доступ к атрибутам объект из вызывающего кода.
- Касательно инкапсуляции непосредственно в языке программирования Python скрыть атрибуты класса можно сделав их приватными или закрытыми и ограничив доступ к ним через специальные методы, которые еще называются **свойствами**.



- **Private.** Приватные члены класса недоступны извне - с ними можно работать только внутри класса.
- **Public.** Публичные методы наоборот - открыты для работы снаружи и, как правило, объявляются публичными сразу по умолчанию.
- **Protected.** Доступ к защищенным ресурсам класса возможен только внутри этого класса и также внутри унаследованных от него классов (иными словами, внутри классов-потомков). Больше никто доступа к ним не имеет.

Для создания приватного атрибута в начале его наименования ставится двойной прочерк: `self.__name`.

К такому атрибуту мы сможем обратиться только из того же класса. Но не сможем обратиться вне этого класса.

Например, присвоение значения этому атрибуту ничего не даст:

```
tom.__age = 43
```

в данном случае просто определяется динамически новый атрибут `__age`, но это он не имеет ничего общего с атрибутом `self.__age`.

Однако все же нам может потребоваться устанавливать возраст пользователя из вне. Для этого создаются свойства. Используя свойство (метод), мы можем получить значение атрибута:

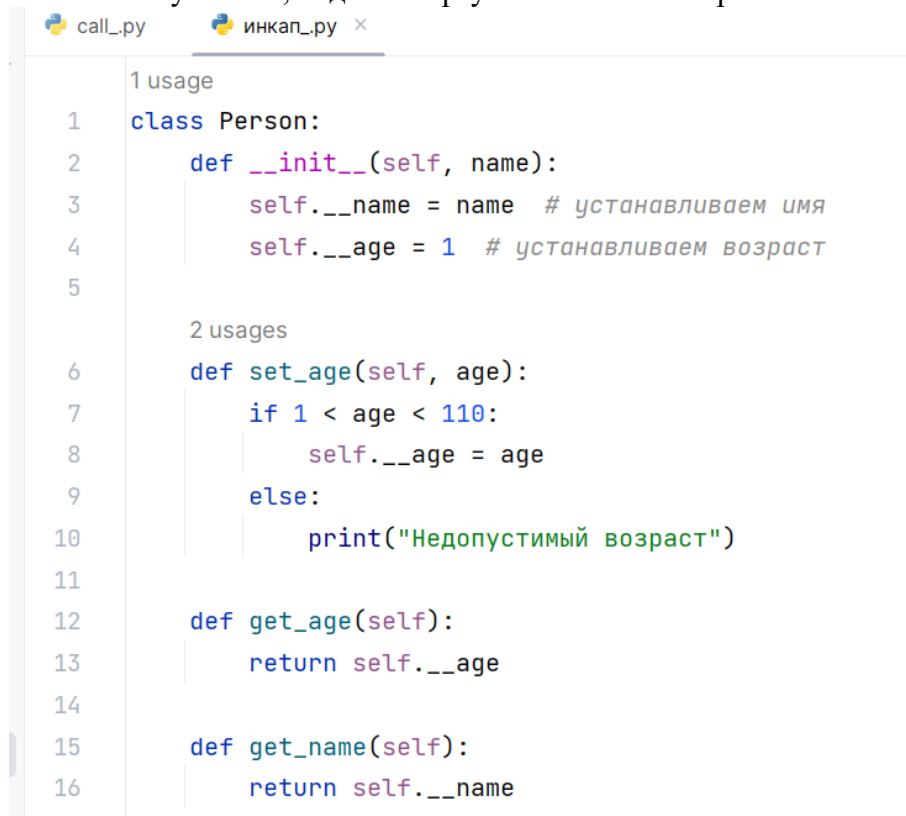
```
def get_age(self):  
    return self.__age
```

Данный метод еще часто называют геттер или аксессор.

Для изменения возраста может быть определено другое свойство:

```
def set_age(self, age):  
    if 1 < age < 110:  
        self.__age = age  
    else:  
        print("Недопустимый возраст")
```

Данный метод еще называют сеттер или мьютейтор (mutator). Здесь мы уже можем решить в зависимости от условий, надо ли переустанавливать возраст.



```
call_py  инкап_py x  
1 usage  
1 class Person:  
2     def __init__(self, name):  
3         self.__name = name # устанавливаем имя  
4         self.__age = 1 # устанавливаем возраст  
5  
2 usages  
6     def set_age(self, age):  
7         if 1 < age < 110:  
8             self.__age = age  
9         else:  
10            print("Недопустимый возраст")  
11  
12    def get_age(self):  
13        return self.__age  
14  
15    def get_name(self):  
16        return self.__name  
--
```

```

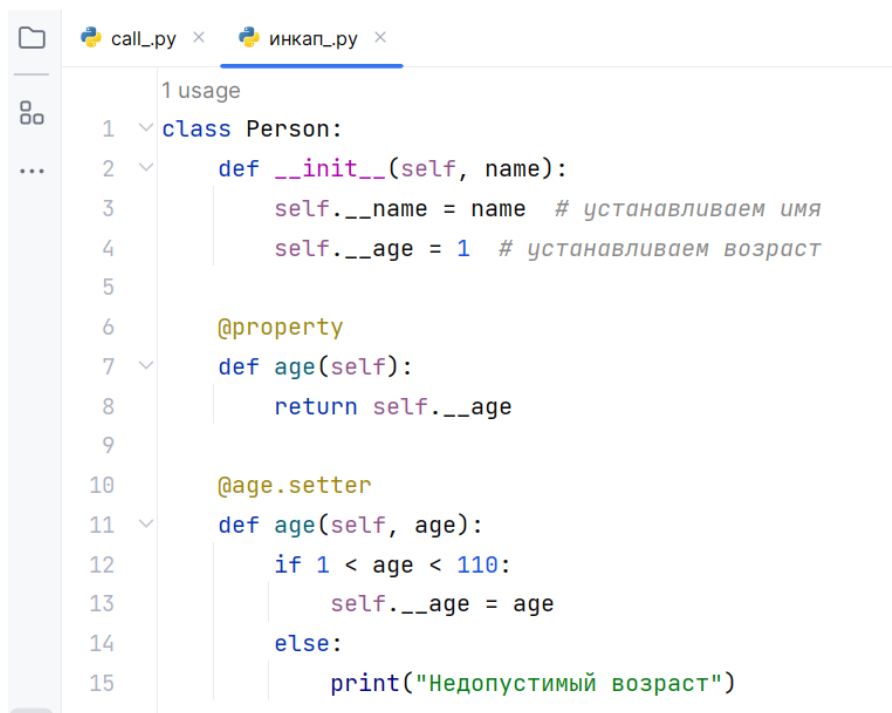
2 usages
18 def display_info(self):
19     print(f"Имя: {self.__name}\tВозраст: {self.__age}")
20
21
22 tom = Person("Том")
23 tom.display_info() # Имя: Том Возраст: 1
24 tom.set_age(-3486) # Недопустимый возраст
25 tom.set_age(25)
26 tom.display_info() # Имя: Том Возраст: 25

```

И сеттер, и геттер называются одинаково – age. И поскольку геттер называется age, то над сеттером устанавливается аннотация @age.setter.

После этого, что к геттеру, что к сеттеру, мы обращаемся через выражение tom.age.

- Python имеет также еще один - более элегантный способ определения свойств. Этот способ предполагает использование аннотаций, которые предваряются символом @.
- Для создания свойства-геттера над свойством ставится аннотация **@property**.
- Для создания свойства-сеттера над свойством устанавливается аннотация **имя_свойства_геттера.setter**.
- Свойство-сеттер определяется после свойства-геттера.



```

call_py x  инкап_py x
1 usage
1 class Person:
2     def __init__(self, name):
3         self.__name = name # устанавливаем имя
4         self.__age = 1 # устанавливаем возраст
5
6     @property
7     def age(self):
8         return self.__age
9
10    @age.setter
11    def age(self, age):
12        if 1 < age < 110:
13            self.__age = age
14        else:
15            print("Недопустимый возраст")

```

```

16
17     @property
18     def name(self):
19         return self.__name
20
21     2 usages
22     def display_info(self):
23         print(f"Имя: {self.__name}\tВозраст: {self.__age}")
24
25 tom = Person("Tom")
26
27 tom.display_info() # Имя: Том  Возраст: 1
28 tom.age = -3486 # Недопустимый возраст
29 print(tom.age) # 1
30 tom.age = 36
31 tom.display_info() # Имя: Том  Возраст: 36

```

Принцип 3. Наследование

Наследование – способ создания нового класса на основе уже существующего, при котором класс-потомок заимствует свойства и методы родительского класса и также добавляет собственные.

Для чего нужно наследование?

Дает возможность использовать код повторно.

Способствует быстрой разработке нового ПО на основе уже существующих открытых классов.

Наследование позволяет делать процесс написания кода более простым.

На что обратить внимание?

- Класс-потомок = Свойства и методы родителя + Собственные свойства и методы.
- Класс-потомок автоматически наследует от родительского класса все поля и методы.
- Класс-потомок может дополняться новыми свойствами.
- Класс-потомок может дополняться новыми методами, а также заменять(переопределять) унаследованные методы. Переопределить родительский метод - это как? Это значит, внутри класса потомка есть метод, который совпадает по названию с методом родительского класса, но функционал у него новый - соответствующий потребностям класса-потомка.

ДОМ

СВОЙСТВА

- 1) Тип фундамента
- 2) Материал крыши
- 3) Количество окон
- 4) Количество дверей

МЕТОДЫ

- 1) Построить
- 2) Отремонтировать
- 3) Заселить
- 4) Снести

Частный дом

СВОЙСТВА

- 1) Тип фундамента (УНАСЛЕДОВАНО)
- 2) Материал крыши (УНАСЛЕДОВАНО)
- 3) Количество окон (УНАСЛЕДОВАНО)
- 4) Количество дверей (УНАСЛЕДОВАНО)
- 5) Количество комнат
- 6) Тип отопления
- 7) Наличие огорода

МЕТОДЫ

- 1) Построить (УНАСЛЕДОВАНО)
- 2) Отремонтировать (УНАСЛЕДОВАНО)
- 3) Заселить (УНАСЛЕДОВАНО)
- 4) Снести (УНАСЛЕДОВАНО)
- 5) Изменить фасад
- 6) Утеплить
- 7) Сделать пристройку

Многоэтажный дом

СВОЙСТВА

- 1) Тип фундамента (УНАСЛЕДОВАНО)
- 2) Материал крыши (УНАСЛЕДОВАНО)
- 3) Количество окон (УНАСЛЕДОВАНО)
- 4) Количество дверей (УНАСЛЕДОВАНО)
- 5) Количество квартир
- 6) Количество подъездов
- 7) Наличие коммерческой недвижимости

МЕТОДЫ

- 1) Построить (УНАСЛЕДОВАНО)
- 2) Отремонтировать (УНАСЛЕДОВАНО)
- 3) Заселить (УНАСЛЕДОВАНО)
- 4) Снести (УНАСЛЕДОВАНО)
- 5) Выбрать управляющую компанию
- 6) Организовать собрание жильцов
- 7) Нанять дворника

Синтаксис

```
class <имя_нового_класса>(<имя_родителя>):
```

Рассмотрим класс **Phone** (Телефон), у которого есть одно свойство **is_on** и три метода:

- инициализатор: **__init__()**;
- включение: **turn_on()**;
- звонок: **call()**.

```
1  # Родительский класс
2  class Phone:
3
4      # Инициализатор
5      def __init__(self):
6          self.is_on = False
7
8      # Включаем телефон
9      def turn_on(self):
10         self.is_on = True
11
12     # Если телефон включен, делаем звонок
13     def call(self):
14         if self.is_on:
15             print('Making call...')
16
```

Создадим новый класс – **MobilePhone** (Мобильный телефон).

- Хотя этот класс и новый, но это по-прежнему телефон, а значит – его все так же можно включить и по нему можно позвонить.
- А раз так, то нам нет смысла реализовывать этот функционал заново, а можно просто унаследовать его от класса **Phone**.

```
17     # Унаследованный класс
18     class MobilePhone(Phone):
19
20         # Добавляем новое свойство battery
21         def __init__(self):
22             super().__init__()
23             self.battery = 0
24
25         # Заряжаем телефон на величину переданного значения
26         def charge(self, num):
27             self.battery = num
28             print(f'Charging battery up to ... {self.battery}%')
29
30
31     my_phone = Phone()
32     my_mobile_phone = MobilePhone()
33     my_mobile_phone.turn_on()
34     my_mobile_phone.call()
35     my_mobile_phone.charge(75)
36
```

Проверка наследования с **issubclass** и **isinstance**:

- **issubclass(sub,sup)** проверяет, является ли класс **sub** подклассом класса **sup**.
- **isinstance(obj,Class)** проверяет, является ли объект **obj** экземпляром класса **Class** или его подкласса.

Еще пример:

Создадим базовый (родительский класс):

```
1 usage
2 class Animal:
3     def __init__(self, name):
4         self.name = name
5
6     1 usage
7     def speak(self):
8         return 'Что-то говорит'
9
10 animal = Animal('Животное')
11 print(animal.speak()) # Что-то говорит
```

Дочерние:

```

8  class Dog(Animal):
    1 usage
9  def speak(self):
10     return 'Гав-гав!'
11
12
13     animal = Animal('Животное')
14     print(animal.speak()) # Что-то говорит
15     dog = Dog('Собака')
16     print(dog.speak()) # Гав-гав!

12  class Bird(Animal):
13     def fly(self):
14         return 'Я летаю'
15
16
17     animal = Animal('Животное')
18     print(animal.speak()) # Что-то говорит
19     dog = Dog('Собака')
20     print(dog.speak()) # Гав-гав!
21     bird = Bird('Птица')
22     print(bird.fly()) # Я летаю

```

Множественное наследование:

```

16  class DogBird(Dog, Bird):
17     pass
18
19     animal = Animal('Животное')
20     print(animal.speak()) # Что-то говорит
21     dog = Dog('Собака')
22     print(dog.speak()) # Гав-гав!
23     bird = Bird('Птица')
24     print(bird.fly()) # Я летаю
25     dogbird = DogBird('Собакоптица')
26     print(dogbird.speak()) # Гав-гав!
27     print(dogbird.fly()) # Я летаю

```

Использование `super()` (это встроенная функция, которая используется для вызова метода в родительском классе)

```

21 class Cat(Animal):
22     def __init__(self, name, breed):
23         super().__init__(name) # вызов __init__ родительского класса
24         self.breed = breed
25
26
27 animal = Animal('Животное')
28 print(animal.speak()) # Что-то говорит
29 dog = Dog('Собака')
30 print(dog.speak()) # Гав-гав!
31 bird = Bird('Птица')
32 print(bird.fly()) # Я летаю
33 dogbird = DogBird('Собакоптица')
34 print(dogbird.speak()) # Гав-гав!
35 print(dogbird.fly()) # Я летаю
36 cat = Cat(name: 'Кот', breed: 'Сиамский')
37 print(cat.name) # "Кот"
38 print(cat.breed) # "Сиамский"

```

Проверим принадлежность:

```

40 print(issubclass(Dog, Animal)) # True
41 print(isinstance(dog, Animal)) # True

```

Принцип 4. Полиморфизм

- **Полиморфизм** – это поддержка нескольких реализаций на основе общего интерфейса (т.е. с разными объектами можно работать единым образом).
- **Абстрактный метод**(он же виртуальный метод) – это метод класса, реализация для которого отсутствует.

Снова обратимся к классам Дом, Частный дом и Многоквартирный дом.

Предположим, что на этапе написания кода мы еще знаем, какой из домов (частный или многоэтажный) нам предстоит создать, но вот то, что какой-то из них придется строить, мы знаем наверняка.

В такой ситуации поступают следующим образом: в родительском классе (в нашем случае – класс Дом) создают пустой метод(например, метод **Построить()**) и делают его абстрактным.

В классах-потомках создают одноименные методы, но уже с соответствующей реализацией. И это логично, ведь, например, процесс постройки Частного и Многоквартирного дома отличается кардинально.

К примеру, для строительства Многоквартирного дома необходимо задействовать башенный кран, а Частный дом можно построить и собственными силами. При этом данный процесс все равно остается процессом строительства.

В итоге получаем метод с одним и тем же именем, который встречается во всех классах. В родительском – имеем только интерфейс, реализация отсутствует. В классах-

потомках – имеем и интерфейс и реализацию. Причем в отличие от родительского класса реализация в потомках уже становится обязательной.

Теперь мы можем увидеть полиморфизм во всей его красе. Даже не зная, с объектом какого из классов-потомков мы работаем, нам достаточно просто вызвать метод Построить(). А уже в момент исполнения программы, когда класс объекта станет известен, будет вызвана необходимая реализация метода Построить().



Пример: пусть есть 2 класса: прямоугольник и квадрат

```
1 class Rectangle:
2     def __init__(self, a, b):
3         self.a = a
4         self.b = b
5
6     def get_rect_pr(self):
7         return 2*(self.a+self.b)
8
9 class Square:
10    def __init__(self, a):
11        self.a = a
12
13    def get_sq_pr(self):
14        return 4*self.a
15
16 r1 = Rectangle(a: 1, b: 2)
17 r2 = Rectangle(a: 6, b: 5)
18 s1 = Square(2)
19 s2 = Square(3)
20 print(r1.get_rect_pr(), r2.get_rect_pr())
21 print(s1.get_sq_pr(), s2.get_sq_pr())
22
```

```

16     r1 = Rectangle(a: 1, b: 2)
17     r2 = Rectangle(a: 6, b: 5)
18     s1 = Square(2)
19     s2 = Square(3)
20
21     geom = [r1, r2, s1, s2]
22     for i in geom:
23         print(i.get_rect_pr())
24

```

```

Run полим x
C:\Users\ludag\PycharmProjects\pythonProject5\Scripts\python.exe C
6
22
Traceback (most recent call last):
  File "C:\Users\ludag\PycharmProjects\pythonProject5\поли.м.py", l
    print(i.get_rect_pr())
AttributeError: 'Square' object has no attribute 'get_rect_pr'

```

Можно решить данную проблему проверкой принадлежности класса:

```

21     geom = [r1, r2, s1, s2]
22     for i in geom:
23         if isinstance(i, Rectangle):
24             print(i.get_rect_pr())
25         else:
26             print(i.get_sq_pr())
27

```

Добавим еще один класс, создадим его экземпляры и добавим их в коллекцию (список):

```

16     class Triangle:
17         def __init__(self, a, b, c):
18             self.a = a
19             self.b = b
20             self.c = c
21
22         def get_tr_pr(self):
23             return self.a+self.b+self.c
24
29     t1 = Triangle(a: 1, b: 2, c: 3)
30     t2 = Triangle(a: 3, b: 2, c: 5)
31
32     geom = [r1, r2, s1, s2, t1, t2]

```

Как придать программе гибкости, чтобы вычисления были верными? Договоримся в каждом классе называть вычисления периметра одинаково, например, `get_pr`. Изменим код для вывода:

```

32     geom = [r1, r2, s1, s2, t1, t2]
33     for i in geom:
34         print(i.get_pr())
35

```

Метод вызывается у соответствующего объекта, это и есть полиморфизм!

Задания для самостоятельной работы:

Задание 1. Описать работу с классами, согласно варианту. Продемонстрировать работу методов классов.

Вариант	Формулировка задания
1, 8, 11, 21, 28	Написать программу, в которой описана иерархия классов: средство передвижения (велосипед, автомобиль, грузовик). Базовый класс должен иметь поля для хранения средней скорости, названия модели, числа пассажиров, а также методы получения потребления топлива для данного расстояния и вычисления времени движения на заданное расстояние.
2, 9, 12, 22, 29	Написать программу, в которой описана иерархия классов: человек («дошкольник», «школьник», «студент», «работающий»). Базовый класс должен иметь поля для хранения ФИО, возраста, пола, а также методы получения среднего дохода и среднего расхода в денежном эквиваленте.
3, 10, 13, 23, 30	Написать программу, в которой описана иерархия классов: геометрические фигуры (эллипс, квадрат, трапеция). Реализовать методы вычисления площади и периметра фигур.
4, 14, 18, 24	Написать программу, в которой описана иерархия классов: функция от одной переменной (синус, косинус, x^3). Базовый класс должен иметь методы получения значения функции для данного значения переменной, а также собой производной текущего экземпляра.
5, 15, 19, 25	Создайте класс «Мебель» с полями «Марка», «Название», «Цена» и методом для вывода подробной информации о предмете. От класса «Мебель» необходимо унаследовать класс «Стол» с унаследованными полями класса «Мебель» и новыми полями «Спинка» (True/False), «Кол-во ножек» и методом для вывода подробной информации.
6, 16, 20, 26	Разработайте класс Book: Автор, Название, Издательство, Год, Количество страниц. Создайте массив объектов. Выведите: а) список книг заданного автора; б) список книг, выпущенных заданным издательством; в) список книг, выпущенных после заданного года. Дополните 2 дочерними классами родительский, определите для них некоторые методы.
7, 17, 27	Создайте класс «Корреспонденция» с полями «Дата», «Тема», «Отправитель» и методом для вывода подробной информации о предмете. От класса «Корреспонденция» необходимо унаследовать класс «Электронное письмо» и класс «Бумажное письмо». Добавить собственные атрибуты. Продумать методы.