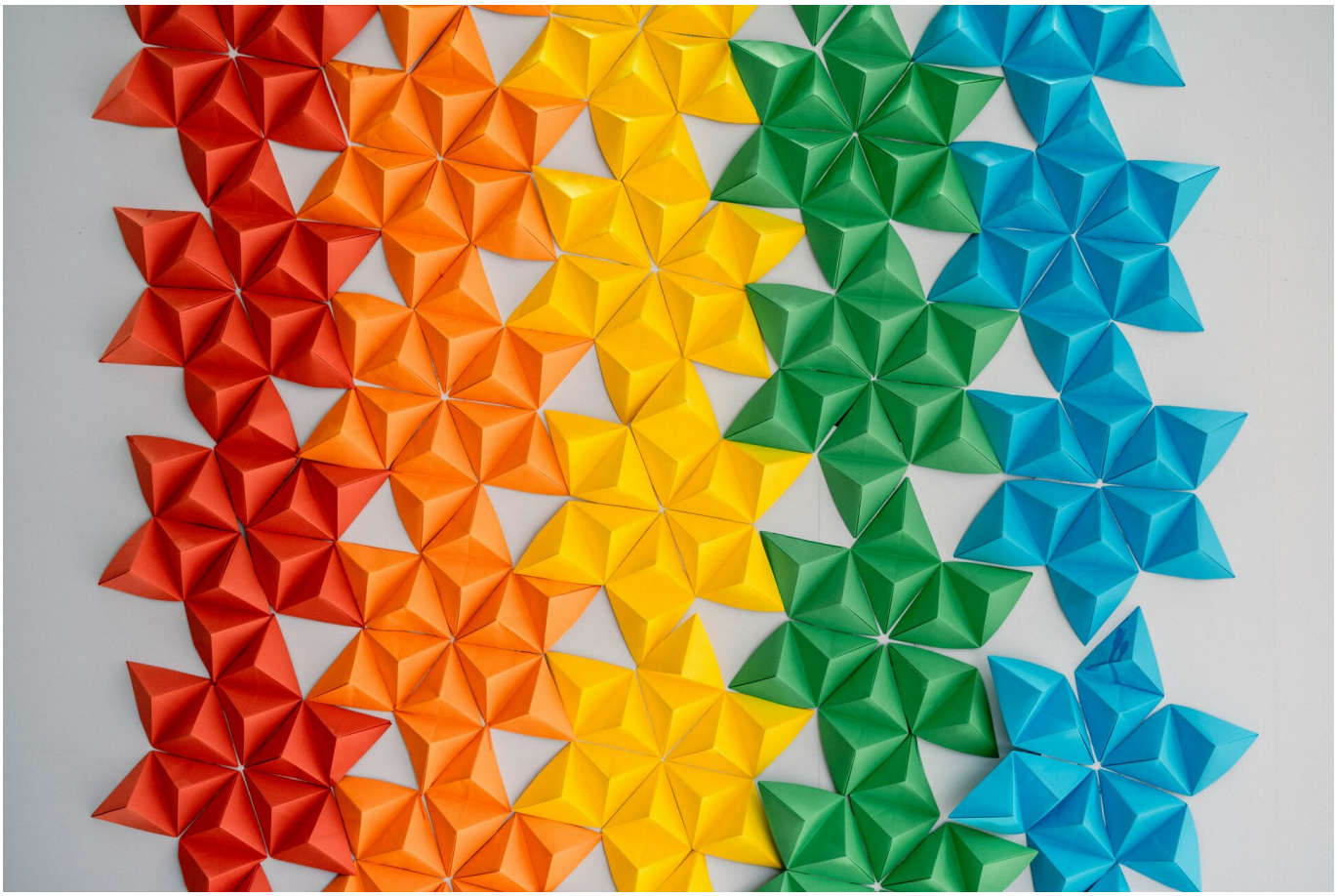


Rekurencja a iteracja w języku Python

- Wprowadzenie
- Przeczytaj
- Gra edukacyjna
- Sprawdź się
- Dla nauczyciela

Źródło: Chinh Le Duc, domena publiczna.

Wiesz już, czym różni się iteracja od rekurencji. Przy rozwiązywaniu wielu problemów, mogą być stosowane zamiennie.

Bywa jednak, że rekurencja błędnie używana jest przy rozwiązywaniu problemów, które nie mają rekurencyjnego charakteru. Zrozumienie wad i zalet rekurencji wymaga umiejętności tworzenia algorytmów w sposób rekurencyjny, dlatego warto przećwiczyć rozwiązywanie problemów zarówno za pomocą rekurencji, jak i iteracji.

W tym e-materiale porównasz metodę iteracyjną oraz rekurencyjną i poznasz wady oraz zalety każdej z nich.

Więcej przykładów, ćwiczeń i rozwiązań znajdziesz w Rekurencja a iteracja w zadaniach.

Ciekawi cię, jak wyglądają implementacje w innych językach programowania? Możesz się z nimi zapoznać w dwóch pozostałych e-materiałach z tej serii:

- Rekurencja a iteracja w języku Java,
- Rekurencja a iteracja w języku C++.

O tym, jak zagadnienie rekurencji wyjaśnia matematyka, przeczytasz w e-materiałach:

- Ciąg określony rekurencyjnie,

- Ciąg geometryczny określony rekurencyjnie,
- Wzór ogólny ciągu określonego rekurencyjnie,
- Ciąg arytmetyczny określony wzorem rekurencyjnym.

Twoje cele

- Zapiszesz algorytm pozwalający obliczyć silnię z wykorzystaniem metody iteracyjnej.
- Przeanalizujesz algorytm wykorzystywany do obliczania silni za pomocą metody rekurencyjnej.
- Prześledzisz ograniczenia wykonywania funkcji rekurencyjnych w języku Python.

Przeczytaj

Wyniki działania dwóch różnych funkcji rozwiązujących ten sam problem są takie same – niezależnie od tego, czy zastosujemy mechanizm rekurencyjny, czy iteracyjny. Odmienny jest natomiast sposób zapisu kodów funkcji oraz sposób ich działania. Przekonamy się o tym na przykładzie dwóch funkcji obliczających wartość silni.

Silnia liczby n jest iloczynem wszystkich liczb naturalnych od 1 do n. Wyjątkiem jest silnia liczby 0, która wynosi 1.

Wzór rekurencyjny na obliczanie $n!$ ma postać:

$$n! = \begin{cases} 1 & dla\ n = 0 \\ n \cdot (n-1) & dla\ n > 0 \end{cases}$$

A oto wzór iteracyjny:

$$n! = 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n$$

Korzystając z przedstawionych wzorów możemy zdefiniować dwie wersje funkcji obliczające wartość silni:

- wykorzystującą mechanizm rekurencji;

Problem 1

- działającą z użyciem metody powtórzeniowej (iteracyjnej):

Problem 2

Problem 3

Dla zainteresowanych

W najpopularniejszej implementacji języka Python – CPython istnieje ograniczenie liczby możliwych do wykonania wywołań rekurencyjnych. Standardowo wynosi ona `1000`. Możemy zmienić tę wielkość, korzystając z funkcji `setrecursionlimit()` należącej do modułu `sys`. Powinniśmy jednak pamiętać, że we wszystkich systemach operacyjnych istnieją fizyczne ograniczenia liczby wykonywanych operacji rekurencyjnych. Wynikają one z ograniczeń pamięci przydzielanych dla stosu w różnych kompilatorach i systemach operacyjnych. Oto przykład wywołania zdefiniowanej wcześniej funkcji `rek_silnia()` zakończony wygenerowaniem informacji o błędzie (później następuje zmiana limitu operacji, dzięki czemu funkcja zwraca poprawny wynik):

```
# wykonanie funkcji bez modyfikacji limitu
rek_silnia(993)

Traceback (most recent call last):
    File "<pyshell#8>", line 1, in <module>
        rek_silnia(993)
    File "/home/python/rek_silnia.py", line 5, in rek_silnia
        return n * rek_silnia(n - 1)
    File "/home/python/rek_silnia.py", line 5, in rek_silnia
        return n * rek_silnia(n - 1)
    File "/home/python/rek_silnia.py", line 5, in rek_silnia
        return n * rek_silnia(n - 1)
    [Previous line repeated 990 more times]
    File "/home/python/rek_silnia.py", line 2, in rek_silnia
        if n == 0:
RecursionError: maximum recursion depth exceeded in comparison

# teraz modyfikujemy limit
import sys
sys.setrecursionlimit(1100)

# ponownie wykonujemy funkcję
rek_silnia(993)
```

```
24
25 # poniżej wynik
26 41094550152463707882643359706479714100653648481957742920230734
27 65066556496369796368412759294613889016590873277970150845487298
28 17158024071203880058200537684026096218297707288997197013835667
29 94464711728415130348167271010608093133244150465522321811685724
30 74878275583576643738093441843763702814070441023918017818834659
31 58823083235484026479844738330233445537220522592228982958851955
32 77612847010697389809177188881268002872716215806214315708365914
33 24336092441448059514822655228783572664718083465069170001421083
34 49219973212394928212222881888040050763475324144752816654627577
35 88586397966169681266522184111950824443472924881515159205715906
36 65988553584288634837740713610288429802630883084653093460045493
37 23272738475581368293184255509340583187468479979440857683863571
38 62999968877386081047258260026271540448070646620665786823786599
39 19779384706214047527454174033210703917725941514020256094434641
40 43681132324872061231869016681390833114207029010669956489697406
41 93018508777075927220539116899954640534695858446258479069732365
42 31972346323784272623701154519991515308745303995920993973977471
43 75560022769919608909070224520953236960384864271520748866061792
44 29684319628789069200620552461352914998454443627642679497852539
45 22125378886775580483987643080481762900065702457032810691517269
46 77550913571570006736204438900067122143753423687161744776165095
47 96899292570863029647005841471056603412621402546607609440523148
48 92300322816810381702204009342830936361518882676186913616959688
49 75418361652474542824023363987570116596008770072060194320460445
50 00000000000000000000000000000000000000000000000000000000000000
51 00000000000000000000000000000000000000000000000000000000000000
52 000000000000000000000000000000000000000000000000
```

Dla zainteresowanych

Spróbujmy określić maksymalną liczbę operacji rekurencyjnych, które można wykonać w naszym systemie operacyjnym. Oto przykładowy program sprawdzający tę wielkość:

```
# kod funkcji
# program domyślnie dla Python 2 dostępny w internecie
# poniżej kod przygotowany dla Python 3

import sys
import itertools

class RecursiveBlowup1:
    def __init__(self):
        self.__init__()

def test_init():
    return RecursiveBlowup1()

class RecursiveBlowup2:
    def __repr__(self):
        return repr(self)

def test_repr():
    return repr(RecursiveBlowup2())

class RecursiveBlowup4:
    def __add__(self, x):
        return x + self

def test_add():
    return RecursiveBlowup4() + RecursiveBlowup4()

class RecursiveBlowup5:
    def __getattr__(self, attr):
        return getattr(self, attr)

def test_getattr():
    return RecursiveBlowup5().attr

```

```
class RecursiveBlowup6:
    def __getitem__(self, item):
        return self[item - 2] + self[item - 1]

def test_getitem():
    return RecursiveBlowup6()[5]

def test_recurse():
    return test_recurse()

def test_cpickle(_cache={}):
    try:
        import _pickle as cPickle
    except ImportError:
        print("cannot import cPickle, skipped!")
        return

    l = None

    for n in itertools.count():
        try:
            l = _cache[n]
            continue  # Already tried and it works, let's sav
        except KeyError:
            for i in range(100):
                l = [l]

        cPickle.dumps(l, protocol=-1)
        _cache[n] = l

def check_limit(n, test_func_name):
    sys.setrecursionlimit(n)

    if test_func_name.startswith("test_"):
        print( test_func_name[5:])
    else:
```

```
72          print(test_func_name)
73
74      test_func = globals()[test_func_name]
75
76      try:
77          test_func()
78      # AttributeError can be raised because of the way e.g. Py
79      # silences all exceptions and returns NULL, which is usua
80      # as "missing attribute".
81      except (RuntimeError, AttributeError):
82          pass
83      else:
84          print( "Yikes!")
85
86  limit = 1000
87
88  while 1:
89      check_limit(limit, "test_recurse")
90      check_limit(limit, "test_add")
91      check_limit(limit, "test_repr")
92      check_limit(limit, "test_init")
93      check_limit(limit, "test_getattr")
94      check_limit(limit, "test_getitem")
95      check_limit(limit, "test_cpickle")
96      print ("Limit of %d is fine" % limit)
97      limit = limit + 100
98
99  # wynik działania dla systemu Linux 64-bity
100 # [...]
101 recurse
102 add
103 repr
104 init
105 getattr
106 getitem
107 cpickle
```

```
Limit of 16800 is fine
recurse

Process finished with exit code 139 (interrupted by signal 1
```

Już wiesz

Podsumujmy najważniejsze elementy z tego e-materiału:

- funkcja rekurencyjna wielokrotnie wywołuje samą siebie,
- wykonanie funkcji wywoływanej rekurencyjnie wymaga większej ilości pamięci operacyjnej,
- w języku Python istnieje limit możliwych do wykonania operacji rekurencyjnych.

Słownik

iteracja

wielokrotne wykonywanie tych samych czynności

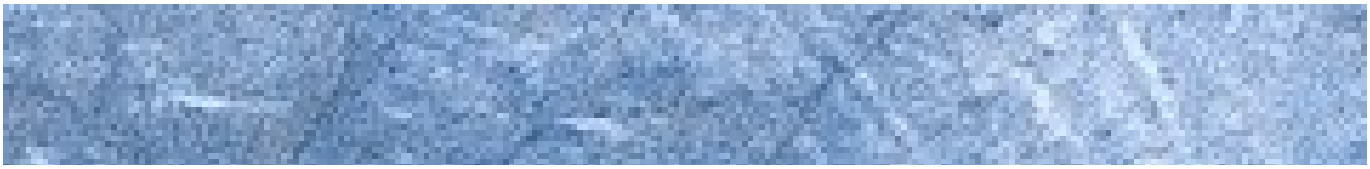
rekurencja

sytuacja, w której funkcja wywołuje samą siebie aż do napotkania przypadku podstawowego

Gra edukacyjna

Polecenie 1

Rozwiąż interaktywny quiz i sprawdź swoją wiedzę.



Test

Rekurencja a iteracja w języku Python --- quiz

Poziom trudności:

InteractiveTest.difficultyLevel.easy

Limit czasu:

8 min

Twój ostatni wynik:

-

Trwa wczytywanie...

Polecenie 2

Sprawdź się

Pokaż ćwiczenia:

Ćwiczenie 1

Napisz program, który metodą iteracyjną wypisze wszystkie liczby naturalne dodatnie nie większe od podanej liczby n i podzielne przez 3 (w kolejności malejącej).
Przetestuj działanie programu dla n = 18.

Specyfikacja:

Dane:

- n – liczba naturalna dodatnia

Wynik:

Program wypisuje ciąg liczb naturalnych dodatnich, spełniających podane warunki (w kolejności malejącej).

Ćwiczenie 2

Napisz program, który metodą rekurencyjną wypisze wszystkie liczby naturalne dodatnie nie większe od podanej liczby n i podzielne przez 2 (w kolejności malejącej). Przetestuj działanie programu dla n = 18.

Specyfikacja problemu:

Dane:

- n – liczba naturalna dodatnia

Wynik:

Program wypisuje wszystkie liczby naturalne dodatnie podzielne przez 2, które są nie większe niż podana liczba. Program wypisuje liczby w kolejności malejącej.

Ćwiczenie 3

Silniowy system pozycyjny jest pozycyjnym sposobem zapisu liczb naturalnych, w którym mnożniki kolejnych pozycji są definiowane przez silnie kolejnych liczb naturalnych, przykładowo:

$$(1220)_{(!)} = (1 \cdot 4!) + (2 \cdot 3!) + (2 \cdot 2!) + (0 \cdot 1!) = 24 + 12 + 4 + 0 = 40$$

Oto inny przykład:

$$(543)_{(!)} = (5 \cdot 3!) + (4 \cdot 2!) + (3 \cdot 1!) = 30 + 8 + 3 = 41$$

Zdefiniuj funkcję `testowa(parametr)`, która będzie przeliczać wartość podaną w postaci *silniowej* do postaci dziesiętnej. Przyjmij, że `parametr` nie może składać się z więcej niż 6 cyfr. Dla parametru mniejszego niż 0 lub większego niż 999999 niech funkcja zwraca wartość `False`.

Specyfikacja problemu:

Dane:

- `parametr` – argument funkcji, będący liczbą w systemie silniowym; liczba naturalna dodatnia, maksymalnie sześciocyfrowa

Wynik:

Program wypisuje wartość liczby `parametr` w systemie dziesiętnym lub wartość `False`, gdy liczba `parametr` nie spełnia założeń zadania.

Sprawdź funkcję dla następujących testów:

- czy funkcja dla parametru mniejszego niż 0 lub większego niż 999999 zwraca typ `bool`,
- czy funkcja dla parametru z zakresu <1, 999999> zwraca typ `int`,

- czy funkcja dla parametru `12345678` zwraca wartość `False`,
- czy funkcja dla parametru `-0.001` zwraca wartość `False`,
- czy funkcja dla parametru `1220` zwraca wartość `40`,
- czy funkcja dla parametru `3742` zwraca wartość `124`.

Dla nauczyciela

Autor: Adam Jurkiewicz
Przedmiot: Informatyka

Temat: Rekurencja a iteracja w języku Python

Grupa docelowa:

Szkoła ponadpodstawowa, liceum ogólnokształcące, technikum, zakres podstawowy i rozszerzony

Podstawa programowa:

Cele kształcenia – wymagania ogólne

I. Rozumienie, analizowanie i rozwiązywanie problemów na bazie logicznego i abstrakcyjnego myślenia, myślenia algorytmicznego i sposobów reprezentowania informacji.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera oraz innych urządzeń cyfrowych: układanie i programowanie algorytmów, organizowanie, wyszukiwanie i udostępnianie informacji, posługiwanie się aplikacjami komputerowymi.

III. Posługiwanie się komputerem, urządzeniami cyfrowymi i sieciami komputerowymi, w tym: znajomość zasad działania urządzeń cyfrowych i sieci komputerowych oraz wykonywania obliczeń i programów.

Treści nauczania – wymagania szczegółowe

I. Rozumienie, analizowanie i rozwiązywanie problemów.

Zakres podstawowy. Uczeń:

2) stosuje przy rozwiązywaniu problemów z różnych dziedzin algorytmy poznane w szkole podstawowej oraz algorytmy:

e) obliczania wartości elementów ciągu metodą iteracyjną i rekurencyjną, w tym wartości elementów ciągu Fibonacciego.

3) wyróżnia w problemie podproblemy i charakteryzuje: metodę połowienia, stosuje podejście zachłanne i rekurencję;

Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

8) dyskutuje na temat roli myślenia komputacyjnego i jego metod, takich jak: abstrakcja, reprezentacja danych, dekompozycja problemu, redukcja, myślenie rekurencyjne, podejście heurystyczne w rozwiązywaniu problemów z różnych dziedzin.

II. Programowanie i rozwiązywanie problemów z wykorzystaniem komputera i innych urządzeń cyfrowych.

Zakres podstawowy. Uczeń:

1) projektuje i programuje rozwiązania problemów z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, funkcje z parametrami i bez parametrów, testuje poprawność programów dla różnych danych; w szczególności programuje algorytmy z punktu I.2);

I + II. Zakres rozszerzony. Uczeń spełnia wymagania określone dla zakresu podstawowego, a ponadto:

3) objaśnia, a także porównuje podstawowe metody i techniki algorytmiczne oraz struktury danych, wykorzystując przy tym przykłady problemów i algorytmów, w szczególności:

b) rekurencję (do generowania ciągów liczb, potęgowania, sortowania liczb, generowania fraktali),

Kształtowane kompetencje kluczowe:

- kompetencje cyfrowe;
- kompetencje osobiste, społeczne i w zakresie umiejętności uczenia się;
- kompetencje matematyczne oraz kompetencje w zakresie nauk przyrodniczych, technologii i inżynierii.

Cele operacyjne (językiem ucznia):

- Zapiszesz algorytm pozwalający obliczyć silnię z wykorzystaniem metody iteracyjnej.
- Przeanalizujesz algorytm wykorzystywany do obliczania silni za pomocą metody rekurencyjnej.
- Prześledzisz ograniczenia wykonywania funkcji rekurencyjnych w języku Python.

Strategie nauczania:

- konstruktywizm;
- konektywizm.

Metody i techniki nauczania:

- dyskusja;
- rozmowa nauczająca z wykorzystaniem multimedium i ćwiczeń interaktywnych.

Formy pracy:

- praca indywidualna;
- praca w parach;
- praca w grupach;
- praca całego zespołu klasowego.

Środki dydaktyczne:

- komputery z głośnikami, słuchawkami i dostępem do internetu;
- zasoby multimedialne zawarte w e-materiale;
- tablica interaktywna/tablica, pisak/kreda;
- oprogramowanie dla języka Python 3 (lub nowszej wersji), w tym PyCharm lub IDLE.

Przebieg lekcji

Przed lekcją:

1. **Przygotowanie do zajęć.** Nauczyciel loguje się na platformie i udostępnia e-materiał: „Rekurencja a iteracja w języku Python". Nauczyciel prosi uczniów o zapoznanie się z treściami w sekcji „Przeczytaj".

Faza wstępna:

1. **Rozpoznanie wiedzy uczniów.** Nauczyciel prosi chętne lub wybrane osoby o przypomnienie definicji iteracji i rekurencji. Pozostali uczniowie uzupełniają ich wypowiedzi lub przedstawiają swoje propozycje.
2. Nauczyciel wyświetla uczniom temat zajęć oraz cele. Prosi, by na ich podstawie uczniowie sformułowali kryteria sukcesu.

Faza realizacyjna:

1. Uczniowie w parach przygotowują przykłady problemów rozwiązanych rekurencyjnie oraz iteracyjnie. Chętne lub wybrane osoby przedstawiają swoje rozwiązania i omawiają je. Nauczyciel prosi o ocenę, które rozwiązanie wydaje im się optymalniejsze.
2. **Ćwiczenie umiejętności.** Uczniowie, pracując w parach, wykonują ćwiczenie nr 1 z sekcji „Sprawdź się". Nauczyciel sprawdza poprawność pisanych kodów, porównuje je i omawia wraz z uczniami. Wskazuje najbardziej efektywne rozwiązanie.

Faza podsumowująca:

1. Nauczyciel ponownie wyświetla na tablicy temat lekcji zawarty w sekcji „Wprowadzenie" i inicjuje krótką rozmowę na temat zrealizowanych celów (czego

uczniowie się nauczyli).
2. Nauczyciel prosi uczniów o podsumowanie zgromadzonej wiedzy w zakresie programowania w języku Python.

Praca domowa:

1. Uczniowie wykonują ćwiczenie nr 1 z sekcji „Gra edukacyjna".

Materiały pomocnicze:

- Oficjalna dokumentacja techniczna dla języka Python 3 (lub nowszej wersji).
- Oficjalna dokumentacja techniczna dla oprogramowania PyCharm lub IDLE.

Wskazówki metodyczne:

- Gra edukacyjna może zostać wykorzystana do przeprowadzenia w klasie ligi zadaniowej.