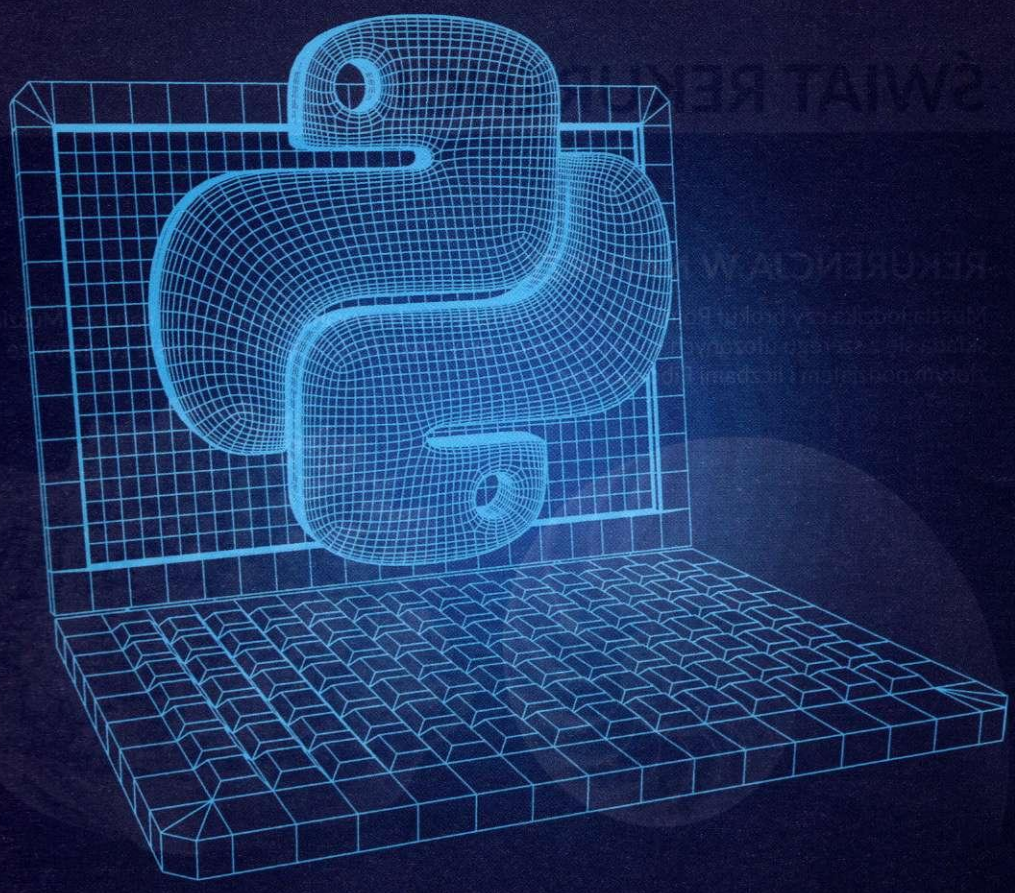


ALGORYTMIKA I PROGRAMOWANIE W PYTHONIE

- Algorytmy zamiany reprezentacji liczb między systemami liczbowymi
- Algorytm połowienia
- Programowanie fraktali
- Iteracja i rekurencja w praktyce
- Programowanie gry

ŚWIAT REKURENCJI

REKURENCJA W NATURZE

Muszla łodzika czy brokuł Romanesco to przykłady występowania rekurencji w naturze. Muszla składa się z szeregu ułożonych spiralnie komór, przy czym kształt spirali jest ściśle związany ze złotym podziałem i liczbami Fibonacciego.

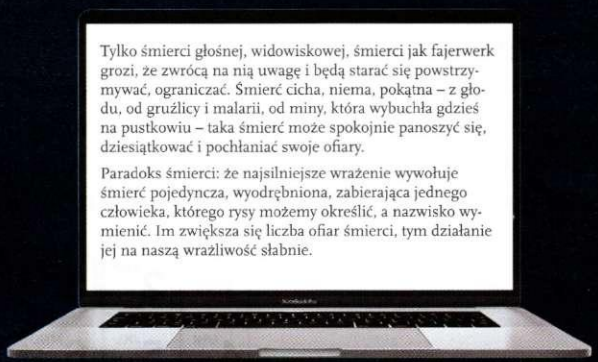
REKURENCJA W GRAFICE I FOTOGRAFII

Rekurencja jest często wykorzystywana w sztukach plastycznych. W fotografii efekt rekurencji ma postać „nieskończonego" powtarzania się malejącego odbicia.

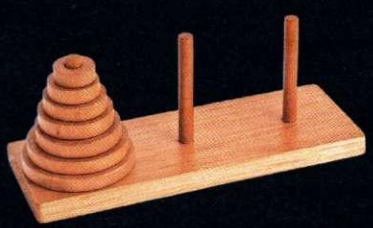
Rekurencja oznacza wywołanie przez funkcję samej na siebie w celu rozwiązania pewnego problemu. Jeśli się zastanowić, to cały świat jest w pewnej mierze rekurencyjny – bo jak inaczej opisać zmywanie naczyń, jedzenie zupy, przechodzenie po pasach przez ulicę, zbieranie rozsypanych kartek...

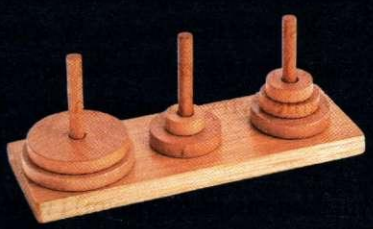
REKURENCJA LINGWISTYCZNA

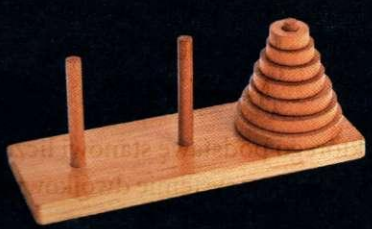
W języku naturalnym rekurencja występuje na różnych poziomach. Rekurencja polegająca na powtórzeniu tego samego słowa lub wyrażenia może mieć na celu maksymalne uspójnienie tekstu albo pełnić funkcję ekspresywną.



Rekurencja w *Lapidarium III* Ryszarda Kapuścińskiego







REKURENCYJNE ŁAMIGŁÓWKI

Na podstawce znajdują się trzy słupki; na jednym z nich umieszcza się osiem krążków o coraz mniejszych średnicach. Należy przenieść wszystkie krążki na inny słupek z zachowaniem następujących reguł:

- wolno przenosić po jednym krążku,
- nie wolno położyć większego krążka na mniejszym,
- można wykorzystywać wszystkie trzy słupki.

Kluczem jest zastosowanie rekurencji.

1. Rozwiąż rekurencyjnie podproblem przenoszenia krążków od 1 do 7 z jakiegokolwiek słupka na dowolny słupek tymczasowy.
2. Przenieś 8. krążek na słupek docelowy.
3. Rozwiąż rekurencyjnie podproblem przenoszenia krążków od 1 do 7 ze słupka tymczasowego na słupek docelowy.

2. Pozycyjne systemy liczbowe

- Zapisywanie liczb w różnych systemach
- Przeliczanie liczb z systemu dwójkowego na dziesiątkowy
- Przeliczanie liczb z systemu dziesiątkowego na dwójkowy

ZADANIE NA START

Alfabet Braille'a

Zarówno liczby, jak i litery można zapisywać za pomocą różnych znaków, np. alfabetu Braille'a, który umożliwia zapisywanie i odczytywanie informacji osobom niewidomym. Gdzie znajdziesz informacje zapisane tym alfabetem? Odszukaj przykład informacji zapisanej tym alfabetem i spróbuj ją odczytać.

SYSTEM BINARNY

Cyfry to umowne znaki graficzne służące do zapisu liczb. Powszechnie stosuje się znaki arabskie i rzymskie, niewidomi posługują się znakami alfabetu Braille'a. Zbiór reguł dotyczących zapisu i nazewnictwa liczb nosi nazwę systemu liczbowego. W systemach o charakterze niepozycyjnym znaczenie cyfry (znaku) jest niezależne od pozycji zajmowanej w liczbie. W systemach pozycyjnych znaczenie cyfry (znaku) zależy od pozycji zajmowanej w liczbie.

Poniżej przedstawiono liczby zapisane w systemie rzymskim (niepozycyjnym) i w powszechnie dziś stosowanym systemie dziesiątkowym (pozycyjnym):

- V = 5,
- VI = 6,
- X = 10,
- XX = 20,
- XXV = 25,
- L = 50,
- C = 100,
- M = 1000,
- MXXV = 1025.

W systemie dziesiątkowym (inaczej: decymalnym), którego podstawę stanowi liczba 10, do zapisu liczb stosuje się 10 cyfr (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), w systemie dwójkowym

(inaczej: binarnym), którego podstawę stanowi liczba 2, do zapisu są wykorzystywane dwie cyfry – 0 i 1.

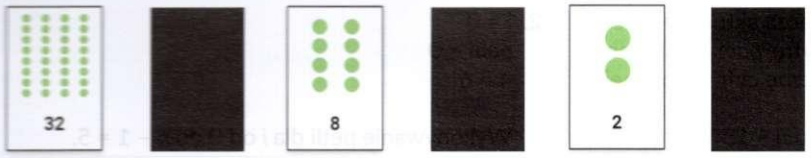
WIEM WIĘCEJ

Niedziesiątkowe systemy zapisu liczb stosuje się m.in. przy pomiarze czasu. Doba ma 24 godziny, godzina – 60 minut, a minuta – 60 sekund.

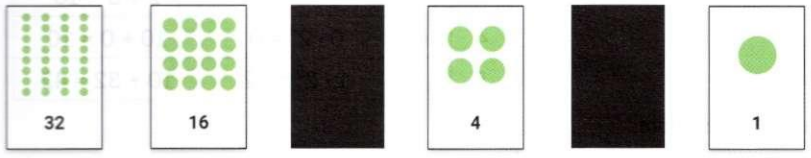
Aby zrozumieć istotę systemu binarnego i kodowania liczb, można wykorzystać karty binarne. Na każdej karcie znajduje się inna liczba kropek, od prawej do lewej: 1, 2, 4, 8, … . Są to kolejne potęgi liczby 2: 2^0, 2^1, 2^2, 2^3, … . Karta może być odwrócona przodem – wtedy widać kropki – lub tyłem – wtedy jest cała czarna, co oznacza 0.

Rys. 1. Karty binarne

Liczby koduje się poprzez łączną liczbę kropek na kartach. Odsłoniętej karcie nadaje się wagę 1, zasłoniętej – 0.



32	0	8	0	2	0	= 42
$1 \cdot 2^5$	$0 \cdot 2^4$	$1 \cdot 2^3$	$0 \cdot 2^2$	$1 \cdot 2^1$	$0 \cdot 2^0$	
1	0	1	0	1	0	= 101010_2



32	16	0	4	0	1	= 53
$1 \cdot 2^5$	$1 \cdot 2^4$	$0 \cdot 2^3$	$1 \cdot 2^2$	$0 \cdot 2^1$	$1 \cdot 2^0$	
1	1	0	1	0	1	= 110101_2

Rys. 2. Liczby 42 i 53 zakodowane za pomocą kart binarnych

Liczba 101010_2 jest zapisem w systemie dwójkowym liczby 42, natomiast liczba 110101_2 jest zapisem w systemie dwójkowym liczby 53. Zestawienie pierwszych 20 liczb w systemie dziesiątkowym i binarnym przedstawiono w poniższej tabeli.

Liczba w systemie dziesiątkowym	Liczba w systemie binarnym	Liczba w systemie dziesiątkowym	Liczba w systemie binarnym
1	1	11	1011
2	10	12	1100
3	11	13	1101
4	100	14	1110
5	101	15	1111
6	110	16	10000
7	111	17	10001
8	1000	18	10010
9	1001	19	10011
10	1010	20	10100

Algorytm zamiany liczby zapisanej dwójkowo na liczbę dziesiętną można przedstawić następująco:

1. Podaj liczbę w systemie dwójkowym.
2. Niech *i* oznacza aktualnie przetwarzaną cyfrę, *pom* – obliczoną sumę, a *n* liczbę cyfr w podanej liczbie.
3. Dla *i* od 0 do *n* – 1:
   - do *x* przypisz *i*-tą od końca cyfrę podanej liczby (liczy się od 0);
   - zwiększ *pom* o wartość $x \cdot 2^i$.
4. Wynikiem jest wartość *pom*.

1. 101010
2. $i = 0$
   $pom = 0$
   $n = 6$
3. Wykonywanie pętli dla *i* od 0 do $n - 1 = 5$.

i	*x*	wartość $x \cdot 2^i$	zwiększanie *pom*
0	0	$0 \cdot 2^0 = 0$	$0 + 0 = 0$
1	1	$1 \cdot 2^1 = 2$	$0 + 2 = 2$
2	0	$0 \cdot 2^2 = 0$	$2 + 0 = 2$
3	1	$1 \cdot 2^3 = 8$	$2 + 8 = 10$
4	0	$0 \cdot 2^4 = 0$	$10 + 0 = 10$
5	1	$1 \cdot 2^5 = 32$	$10 + 32 = 42$

4. 42

W komputerowej realizacji algorytmu należy zwrócić uwagę na typy danych. O ile wartości w systemie dziesiątkowym są zapisywane jako liczba, o tyle w systemie dwójkowym traktuje się je jako napis.

Ćwiczenie 1. Z systemu dwójkowego na dziesiątkowy

Przeanalizuj dane w tabeli i zdefiniuj funkcję **na10(dana)**, której parametrem jest liczba binarna w postaci napisu, a wynikiem – odpowiadająca jej liczba w systemie dziesiątkowym.

Wywołanie funkcji	Wynik
`na10("101010")`	42
`na10("110101")`	53

- Przeanalizuj przykłady z zastosowaniem opisanego wyżej algorytmu. Dla wartości **101010** algorytm został już rozpisany. W ten sam sposób rozpisz algorytm dla **110101**.
- Przypisz zmiennej pomocniczej wartość początkową i zapisz szkielet pętli **for**. Ponieważ dane mają postać napisów, do znalezienia długości liczby binarnej możesz wykorzystać funkcję **len()**. Aby uniknąć ewentualnych błędów, zostaw miejsce przeznaczone na polecenia wykonywane w pętli – wpisz słowo kluczowe **pass**.

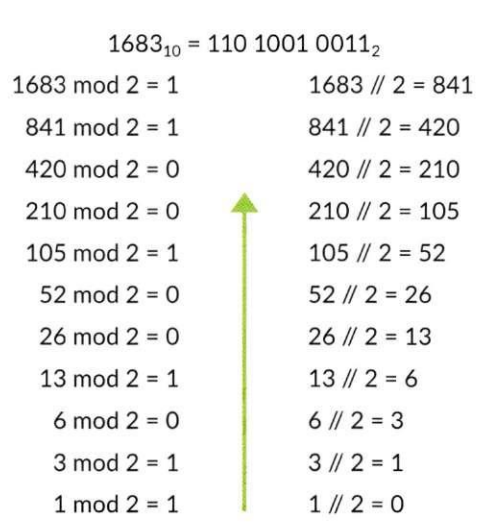
```
1. def na10(dana):
2.     pom = 0
3.     for i in range(len(dana)):
4.         pass
5.     return pom
```

- Uzupełnij zapis. Zmienna **pom** na początku przyjmuje wartość 0. Każdą cyfrę należy zamienić na liczbę całkowitą za pomocą funkcji **int()**, a następnie pomnożyć przez kolejną potęgę liczby 2 i dodać do bieżącej wartości **pom**.

```
1. def na10(dana):
2.     pom = 0
3.     for i in range(len(dana)):
4.         x = int(dana[- i - 1])
5.         pom = pom + x * 2 ** i
6.     return pom
```

- Sprawdź działanie skryptu z różnymi danymi.
  - Jeśli pracujesz w edytorze online, wypisz wynik funkcji, np. **print(na10("101010"))**.
  - Jeśli pracujesz w pliku, wywołaj funkcję **na10("101010")**.

Odwrotny proces – zamiany liczby dziesiętnej na binarną – wymaga zastosowania operacji reszty dzielenia (mod) przez 2 do wyodrębnienia każdorazowo ostatniej cyfry oraz dzielenia całkowitego przez 2. Obliczenia należy kontynuować tak długo, dopóki liczba jest większa od 0.



Rys. 3. Zamiana liczby dziesiętnej na binarną

Po przeanalizowaniu algorytmu na przykładzie można zapisać go następująco:

1. Podaj liczbę w systemie dziesiątkowym.
2. Jeżeli *liczba* jest równa 0, to wypisz 0 i zakończ działanie algorytmu.
3. Przypisz do zmiennej *pom* (zapamiętywane kolejne cyfry) napis pusty.
4. Dopóki *liczba* > 0, wykonuj:
   - dopisz do *pom* wartość *liczba* % 2;
   - przypisz do *liczba* wartość *liczba* // 2.
5. Wypisz *pom*.

WIEM WIĘCEJ

W widoku Programisty kalkulatora systemowego Windows istnieje możliwość przeliczania liczb z jednego systemu na drugi oraz przeprowadzania operacji na liczbach.

Ćwiczenie 2. Z systemu dziesiątkowego na dwójkowy

Przeanalizuj dane w tabeli i zdefiniuj funkcję **na2(liczba)**, której parametrem jest liczba dziesiętna, a wynikiem – odpowiadająca jej liczba zapisana jako napis w systemie dwójkowym.

Wywołanie funkcji	Wynik
na2(42)	**"101010"**
na2(53)	**"110101"**

- Przeanalizuj przykłady z zastosowaniem opisanego wyżej algorytmu.
- Zapisz instrukcję warunkową – jeśli liczba jest równa 0, wypisz 0 i zakończ działanie algorytmu.

- Przypisz zmiennej pomocniczej pusty napis.
- W pętli **while** obliczaj resztę z dzielenia liczby przez 2, dopóki **liczba > 0**, i po zamianie na napis za pomocą funkcji **str()** wstawiaj ją na początek zmiennej pomocniczej. Do dalszych obliczeń musisz wziąć wynik całkowitego dzielenia liczby przez 2.

```
1. def na2(liczba):
2.     if liczba == 0:
3.         return 0
4.     pom = ""
5.     while liczba > 0:
6.         pom = str(liczba % 2) + pom
7.         liczba = liczba // 2
8.     return pom
```

- Sprawdź działanie skryptu z różnymi danymi.

POZOSTAŁE SYSTEMY

System dwójkowy wykorzystuje się w informatyce i elektronice cyfrowej, gdzie ograniczenie liczby stanów do dwóch – 0 i 1, tj. niskie napięcie i wysokie napięcie – gwarantuje łatwą implementację techniczną, minimalizuje zakłócenia (przekłamania danych) i daje możliwość interpretacji cyfr jako wartości logicznych. Niemniej jednak system ten nie jest wygodny w zapisie, dlatego też często stosuje się przeliczenie na system ósemkowy lub szesnastkowy – pozwala to skrócić zapis bez konwersji na system dziesiątkowy (np. system szesnastkowy umożliwia czterokrotne skrócenie zapisu liczby binarnej).

POSZUKAJ W INTERNECIE

System szesnastkowy (heksadecymalny) jest stosowany m.in. w programowaniu, adresach MAC, opisywaniu kolorów w językach HTML i CSS oraz w programach graficznych. Znajdź przykłady konkretnych zastosowań.

W systemach mniejszych od dziesiątkowego, a więc trójkowym, czwórkowym, …, dziewiątkowym, liczby są zapisywane odpowiednio za pomocą cyfr: 0, 1 i 2; 0, 1, 2 i 3; …; 0, 1, 2, 3, 4, 5, 6, 7 i 8. Przeliczanie liczb między systemami też jest analogiczne – bierze się pod uwagę podstawę systemu do potęgi i, czyli w konwersji z systemu ósemkowego na dziesiątkowy wartość x mnoży się przez 8^i.

Algorytm można zapisać następująco:

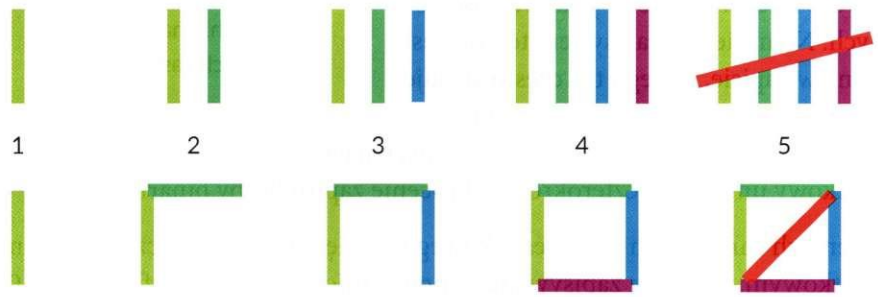
1. Podaj liczbę w systemie ósemkowym.
2. Niech i oznacza aktualnie przetwarzaną cyfrę, *pom* – obliczoną sumę, a n – liczbę cyfr w podanej liczbie.
3. Dla i od 0 do $n - 1$:
   - do x przypisz i-tą od końca cyfrę;
   - zwiększ *pom* o wartość $x \cdot 8^i$;
4. Wynikiem jest wartość *pom*.

Zestawienie liczb w systemie dziesiątkowym i ósemkowym przedstawiono w poniższej tabeli.

Liczba w systemie dziesiątkowym	Liczba w systemie ósemkowym	Liczba w systemie dziesiątkowym	Liczba w systemie ósemkowym
1	1	11	13
2	2	12	14
3	3	13	15
4	4	14	16
5	5	15	17
6	6	16	20
7	7	17	21
8	10	18	22
9	11	19	23
10	12	20	24

WIEM WIĘCEJ

W systemie jedynkowym, który wykorzystuje się do zliczania, występuje jeden znak, np. kreska. Kolejne liczby są tworzone przez powtarzanie tego znaku. Ponieważ już przy trochę większych liczbach staje się to mało czytelne, zazwyczaj grupuje się znaki.



Ćwiczenie 3. Z systemu ósemkowego na dziesiątkowy

Przeanalizuj dane w tabeli i zdefiniuj funkcję **na10(dana)**, której parametrem jest liczba zapisana w systemie ósemkowym, a wynikiem – odpowiadająca jej liczba zapisana w systemie dziesiątkowym.

Wywołanie funkcji	Wynik
na10("174")	**124**
na10("362374")	**124156**

Ćwiczenie 4. Z systemu dziesiątkowego na ósemkowy

Przeanalizuj dane w tabeli i zdefiniuj funkcję **na8(liczba)**, której parametrem jest liczba dziesiętna, a wynikiem – odpowiadająca jej liczba zapisana w systemie ósemkowym.

Wywołanie funkcji	Wynik
na8(124)	**"174"**
na8(124156)	**"362374"**

Systemy liczbowe większe niż dziesiątkowy potrzebują dodatkowych znaków. W systemie szesnastkowym poza cyframi dziesiętnymi używa się pierwszych sześciu liter alfabetu łacińskiego: A, B, C, D, E, F (wielkich lub małych). Cyfry 0–9 mają te same wartości co w systemie dziesiątkowym, natomiast litery odpowiadają wartościom 10–15: A – 10, B – 11, C – 12, D – 13, E – 14 oraz F – 15.

Zestawienie liczb w systemie dziesiątkowym i szesnastkowym przedstawiono w poniższej tabeli.

10	16	10	16	10	16	10	16	10	16	10	16
1	1	11	B	21	15	31	1F	41	29	51	33
2	2	12	C	22	16	32	20	42	2A	52	34
3	3	13	D	23	17	33	21	43	2B	53	35
4	4	14	E	24	18	34	22	44	2C	54	36
5	5	15	F	25	19	35	23	45	2D	55	37
6	6	16	10	26	1A	36	24	46	2E	56	38
7	7	17	11	27	1B	37	25	47	2F	57	39
8	8	18	12	28	1C	38	26	48	30	58	3A
9	9	19	13	29	1D	39	27	49	31	59	3B
10	A	20	14	30	1E	40	28	50	32	60	3C

Ćwiczenie 5. Z systemu szesnastkowego na dziesiątkowy

Przeanalizuj postępowanie dla liczby 6B8E23, a następnie przelicz w ten sam sposób następujące liczby zapisane w systemie szesnastkowym na system dziesiątkowy: FF, CCF3, 339F, 2EAB7.

- Rozpisz liczbę w tabeli.

6	B	8	E	2	3
$6 \cdot 16^5$	$11 \cdot 16^4$	$8 \cdot 16^3$	$14 \cdot 16^2$	$2 \cdot 16^1$	$3 \cdot 16^0$

- Zsumuj zawartość kolejnych komórek.

$6 \cdot 16^5 + 11 \cdot 16^4 + 8 \cdot 16^3 + 14 \cdot 16^2 + 2 \cdot 16^1 + 3 \cdot 16^0 = 7\,048\,739$

Ćwiczenie 6. Z systemu dziesiątkowego na szesnastkowy

Przeanalizuj postępowanie dla trzech liczb: 107, 142, 35, a następnie przelicz w ten sam sposób następujące trójki liczb zapisane w systemie dziesiątkowym na system szesnastkowy: 165 42 42, 189 183 107, 112 128 144, 47 79 79. Znajdź w internecie nazwy odpowiadających im kolorów.

- Dla liczby 107
  107 : 16 = 6 r. 11
  6 : 16 = 0 r. 6, czyli 107 to 6B
- Dla liczby 142
  142 : 16 = 8 r. 14
  8 : 16 = 0 r. 8, czyli 142 to 8E
- Dla liczby 35
  35 : 16 = 2 r. 3
  2 : 16 = 0 r. 2, czyli 35 to 23
- Kolor RGB(107, 142, 35) zapisany w systemie szesnastkowym to #6B8E23, czyli OliveDrab.

WIEM WIĘCEJ

W języku Python można korzystać z gotowych funkcji do konwersji liczb zapisanych w jednym systemie na inny.

- Zamiana na system dwójkowy – jest dopisywany przedrostek 0b: **`bin(231)`**.
- Zamiana na system szesnastkowy – jest dopisywany przedrostek 0x: **`hex(202)`**.
- Zamiana na system dziesiątkowy z systemów dwójkowego, trójkowego i szesnastkowego: **`int("11010", 2)`**; **`int("21100", 3)`**; **`int("AA100", 16)`**.

PODSUMOWANIE

- Każdą liczbę można zapisać w dowolnym systemie pozycyjnym z użyciem określonej liczby cyfr, a w niektórych przypadkach również liter. W systemie dwójkowym są to cyfry 0 i 1, w ósemkowym: 0–7, a w szesnastkowym cyfry 0–9 oraz litery A(10)–F(15).
- System binarny wykorzystuje się w informatyce i elektronice cyfrowej, ponieważ ograniczenie liczby stanów do dwóch – 0 i 1 – pozwala na łatwą implementację techniczną, ogranicza przekłamania danych i umożliwia interpretację cyfr jako wartości logicznych.
- Przy przeliczaniu z systemu dwójkowego na system dziesiątkowy mnoży się kolejne cyfry od końca przez kolejne potęgi dwójki i dodaje iloczyny. Podobnie postępuje się w przypadku innych systemów – np. dla szesnastkowego mnoży się przez potęgi 16 i dodaje iloczyny.

- Podczas przeliczania z systemu dziesiątkowego na inny dzieli się przez podstawę nowego systemu z zapamiętaniem reszt. Reszty czytane od końca tworzą szukaną liczbę.
- W komputerowej realizacji algorytmu konwersji liczb między systemami liczbowymi należy zwrócić uwagę na typy danych, np. w systemie dziesiątkowym są one traktowane jako liczba, a w systemie dwójkowym – jako napis.
- Do znalezienia długości liczby binarnej można wykorzystać funkcję **`len()`**.
- Aby zamienić cyfrę na liczbę całkowitą, należy zastosować funkcję **`int()`**. Aby zamienić liczbę na napis, należy zastosować funkcję **`str()`**.

PYTANIA SPRAWDZAJĄCE

1. Jak rozpoznać, czy liczba zapisana binarnie jest parzysta czy nieparzysta, bez przeliczania na system dziesiątkowy? Przeanalizuj to na przykładach: 100010_2, 1001_2, 11001_2, 10100_2.
2. Czy można jednoznacznie określić, w jakim systemie jest zapisana liczba, gdy widać, z jakich cyfr się składa? W uzasadnieniu podaj przykłady.
3. Jaka liczba jest o jeden większa od 5F w systemie szesnastkowym?

ZADANIA DODATKOWE

Zadanie 1. Bintooc

Napisz funkcję **`bintooct(liczba)`**, której parametrem jest liczba zapisana w systemie dwójkowym, a wynikiem – odpowiadająca jej liczba zapisana w systemie ósemkowym. Postaraj się wykorzystać fakt, że 8 jest potęgą dwójki.

Zadanie 2. Octtobin

Napisz funkcję **`octtobin(liczba)`**, której parametrem jest liczba zapisana w systemie ósemkowym, a wynikiem – odpowiadająca jej liczba zapisana w systemie dwójkowym. Postaraj się wykorzystać fakt, że 8 jest potęgą dwójki.

3. Metoda połowienia

- Zasady działania algorytmu połowienia
- Operacje na liczbach zmiennoprzecinkowych
- Implementacja i zastosowania algorytmu połowienia

ZADANIE NA START

Gra w zgadywanie liczby

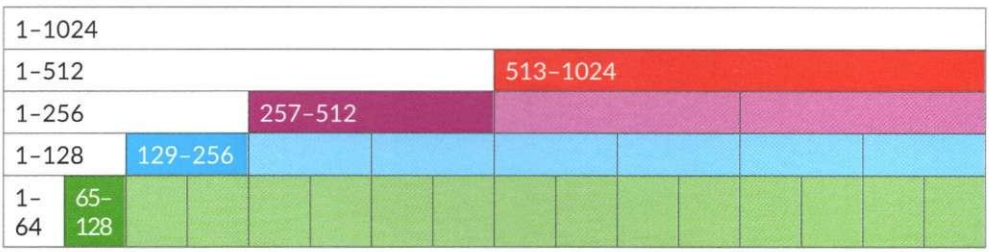
Ania z Wojtkiem i Marcinem grają w zgadywanie liczby. Ania wymyśla liczbę z przedziału 1–1024, a chłopcy zgadują. Wojtek kolejno pyta, czy to jest 1, 2, 3 ..., a Marcin postępuje według strategii wyszukiwania binarnego. Najpierw pyta o 1024/2 = 512 i w zależności od tego, jak brzmi odpowiedź: czy ta liczba jest mniejsza czy większa, rozpatruje kolejny przedział. Jeśli mniejsza, to połowi przedział 1–512, a jeśli większa – przedział 513–1024. W ilu krokach na pewno zgadnie Wojtek, a w ilu – Marcin?

O METODZIE POŁOWIENIA

Ogólna zasada działania algorytmu połowienia polega na dzieleniu zakresu liczb na połowę i sprawdzaniu, czy element znajdujący się dokładnie pośrodku jest elementem szukanym, a jeśli nie, to czy jest większy, czy mniejszy od szukanego, oraz na przeszukiwaniu przedziału odpowiednio na prawo lub na lewo. W ten sposób bardzo szybko zawęża się zakres poszukiwań (w każdym kroku o połowę), aż w końcu otrzymuje się przedział jednoelementowy. Można wtedy wskazać szukany element lub stwierdzić, że go nie ma.

POSZUKAJ W INTERNECIE

Algorytm połowienia ma szersze zastosowania niż tylko wyszukiwanie liczby w zbiorze uporządkowanym. Znajdź w sieci inne przykłady wykorzystania tej metody wyszukiwania.



Rys. 1. Kolejne kroki algorytmu – początkowe zakresy przedziałów dla gry w zgadywanie liczby

WIEM WIĘCEJ

Wyszukiwanie metodą połowienia jest znacznie szybsze niż wyszukiwanie liniowe. Funkcja matematyczna opisująca zależność rozmiaru danych od liczby kroków, w którym otrzymuje się odpowiedź, to logarytm o podstawie 2: $\log_2 1024 = 10$ (bo $2^{10} = 1024$), $\log_2 512 = 9$ (bo $2^9 = 512$), $\log_2 256 = \ldots$.

ZNAJDOWANIE PIERWIASTKA Z LICZBY Z ZADANYM PRZYBLIŻENIEM

Jak znaleźć wartość $\sqrt{2}$ z dokładnością 0,001, jeśli nie ma gotowej funkcji **pierwiastek**? Albo inaczej: jak znaleźć wartość pierwiastka kwadratowego podanej liczby $x \geq 1$ z przybliżeniem *eps*, jeśli można korzystać tylko z dodawania, odejmowania, mnożenia i dzielenia?

Przybliżenie służy do określania dokładności, z jaką oblicza się pierwiastek. Powinno być stosunkowo małą liczbą. Wobec tego szukana jest taka liczba a, dla której prawdziwa jest nierówność $|\sqrt{x} - a| < eps$.

Podczas formułowania algorytmu warto wykorzystać metodę połowienia. Należy zacząć od ustalenia dwóch końców przedziałów. Dopóki długość przedziału jest większa niż przybliżenie, trzeba znajdować środek przedziału i zawężać go do lewej części lub prawej. Można zapisać to następująco:

1. Podaj x i *eps*.
2. Ustal końce przedziału $a = 0$ i $b = x$.
3. Dopóki $|b - a| > eps$, wykonuj:
   - oblicz $c = (a + b) / 2$;
   - jeżeli $c \cdot c > x$, przypisz $b = c$;
   - w przeciwnym wypadku przypisz $a = c$.
4. Wynikiem jest c.

1. $x = 2$, $eps = 0{,}001$
2. $a = 0$, $b = 2$
3. Wykonywanie pętli, dopóki $|b - a| > 0{,}001$.

a	b	$\lvert b - a \rvert$	c	
0	2	2	1	nowe a
1	2	1	1,5	nowe b
1	1,5	0,5	1,25	nowe a
1,25	1,5	0,25	1,375	nowe a
1,375	1,5	0,125	1,4375	nowe b
1,375	1,4375	0,0625	1,40625	nowe a
1,40625	1,4375	0,03125	1,421875	nowe b
1,40625	1,421875	0,015625	1,4140625	nowe a
1,4140625	1,421875	0,0078125	1,41796875	nowe b
1,4140625	1,41796875	0,00390625	1,416015625	nowe a
1,4140625	1,416015625	0,001953125	1,4150390625	nowe b
1,4140625	1,4150390625	**0,0009765625**		

4. 0,0009765625 < 0,001, więc wynikiem jest liczba 1,4150390625.

Zapis algorytmu połowienia w Pythonie będzie zbliżony do zapisu przedstawionego w języku naturalnym. Jedyne, czego potrzeba podczas implementacji tej strategii postępowania, to funkcja przekazująca wartość bezwzględną. Jest to funkcja **abs()**, której argumentem może być liczba całkowita lub rzeczywista.

Ćwiczenie 1. Pierwiastek kwadratowy

Napisz funkcję **pierwiastek(x, eps)**, której wynikiem jest $\sqrt{x}$ z dokładnością **eps**. Pierwszym argumentem x jest liczba dodatnia większa lub równa 1, a drugim – mała liczba dodatnia, np. 0,00001.

- Napisz deklarację funkcji. Możesz skorzystać z wartości domyślnej parametrów.

```
1. def pierwiastek(x, eps):
2.     pass
```

- Ustal końce przedziału – zmienne **a** i **b**. Do iteracji wykorzystaj pętlę **while**. Podczas zapisywania poleceń wykonywanych w pętli zwróć uwagę na odpowiednie wcięcia.

```
1. def pierwiastek(x, eps):
2.     a = 0
3.     b = x
4.     while abs(b - a) > eps:
5.         c = (a + b) / 2
6.         if c * c > x:
7.             b = c
8.         else:
9.             a = c
10.    return c
```

- Sprawdź działanie skryptu z różnymi danymi, np. wyznacz $\sqrt{2}$ z dokładnością 0,1, 0,01 oraz 0,00001.

Ćwiczenie 2. Dowolny pierwiastek

Napisz funkcję **pierwiastek(x, n, eps)**, której wynikiem jest $\sqrt[n]{x}$ z dokładnością **eps**. Pierwszym argumentem **x** jest liczba dodatnia większa lub równa 1, drugim – stopień pierwiastka, a trzecim – mała liczba dodatnia, np. 0,00001.

Wskazówka: zmodyfikuj algorytm z ćwiczenia 1 – dodaj dodatkowy parametr **n** i podnieś **c** do **n**-tej potęgi.

WIEM WIĘCEJ

Algorytm znajdowania pierwiastka z liczby z zadanym przybliżeniem jest poprawny tylko dla liczb większych lub równych 1. Dla liczb mniejszych od 1 wchodzi w pętlę nieskończoną.

PROBLEMY Z LICZBAMI RZECZYWISTYMI

Aby przekonać się, jaki problem można napotkać, posługując się liczbami rzeczywistymi, uruchom kod z rys. 2. Na początku wartość zmiennej **x** wynosi **0**, w każdym obrocie pętli jest zwiększana o **0.1**. Czy program się zatrzyma?

```
1. x = 0.0
2. while x != 1.0:
3.     x = x + 0.1
4.     print(x)
```

Rys. 2. Testowanie dokładności operacji dodawania na liczbach rzeczywistych

Program się nie zatrzymuje.

```
0.1
0.2
0.30000000000000004
0.4
0.5
0.6
0.7
0.7999999999999999
0.8999999999999999
0.9999999999999999
1.0999999999999999
1.2
1.3
1.4000000000000001
1.5000000000000002
1.6000000000000003
```

Rys. 3. Efekt działania programu – dokładna wartość `1.0` nie występuje

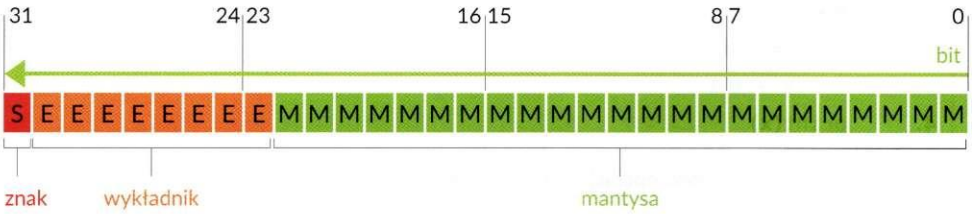
Wystarczy lekko zmodyfikować kod, a wszystko będzie w porządku. Porównywanie wartości `!=` (różne) trzeba zastąpić relacją mniejszości `<`. Nie jest wtedy badany warunek `x == 0.1`, a moment przekroczenia progu 1.

```
1. x = 0.0
2. while x < 1.0:
3.     x = x + 0.1
4.     print(x)
```

Rys. 4. Poprawny zapis badania warunku przekroczenia progu

Na pierwszy rzut oka wydaje się to mało znaczącą zmianą, jednak istotnie wpływa na działanie algorytmu. Dlaczego? Odpowiedzi należy szukać w sposobie reprezentacji liczb zmiennoprzecinkowych w komputerze.

Każdą liczbę zmiennoprzecinkową zapisuje się za pomocą trzech składowych: znaku (S – *sign*, 1 lub –1), wykładnika (E – *exponent*) i mantysy (M – *mantissa*), czyli części ułamkowej. Podobnie jest w notacji naukowej, np. $1{,}2 \cdot 10^{13}$ (czyli 12 000 000 000 000).



Rys. 5. Reprezentacja liczb zmiennoprzecinkowych w komputerze

Taki sposób przechowywania liczb rzeczywistych pozwala na ich zapamiętywanie z określoną dokładnością. Należy to uwzględnić przy pisaniu programów. Warto o tym pamiętać i sprawdzać równość liczb z określonym przybliżeniem.

ZNAJDOWANIE MIEJSCA ZEROWEGO FUNKCJI

Problem znajdowania pierwiastka jest szczególnym przypadkiem znajdowania miejsca zerowego funkcji. Aby można było zastosować algorytm połowienia dla danej funkcji f w przedziale $\langle a, b\rangle$, muszą zostać spełnione trzy warunki:

- funkcja musi być określona w danym przedziale, tj. tak, żeby można było obliczyć jej wartość w każdym punkcie przedziału (np. funkcja $1/x$ dla $x = 0$ nie jest określona);
- funkcja musi być ciągła;
- wartości funkcji na końcach przedziału $\langle a, b\rangle$ muszą mieć różne znaki.

Za pomocą algorytmu połowienia można znaleźć tylko jedno miejsce zerowe.

Ćwiczenie 3. Miejsce zerowe funkcji

Napisz program, za pomocą którego znajdziesz miejsce zerowe funkcji określonej wzorem $f(x) = 2x^3 - x - 5$ w przedziale $\langle 0, 100\rangle$ z dokładnością 0,001. Wykorzystaj algorytm połowienia.

- Sprawdź, czy funkcja przyjmuje różne wartości na końcach przedziału. Przyjmij końce przedziału jako wartości początkowe.
  $f(0) = 2 * 0^3 - 0 - 5 = -5$ (liczba ujemna)
  $f(100) = 2 * 100^3 - 100 - 5 = 1000000 - 100 - 5 = 1999895$ (liczba dodatnia)
- Napisz funkcję pomocniczą $f(x)$, której wynikiem będzie wartość funkcji dla argumentu x.

```
1. def f(x):
2.     return 2 * x ** 3 - x - 5
```

- Zaimplementuj algorytm z użyciem podanych wartości. Skorzystaj z wbudowanej funkcji **abs()**. Zwróć uwagę, że podnoszenie liczby c do potęgi zastąpiono ogólnym zapisem $f(c)$.

```
1. def f(x):
2.     return 2 * x ** 3 - x - 5
3.
4. def miejsce_zerowe(eps):
5.     a = 0
6.     b = 100
7.     while abs(b - a) > eps:
8.         c = (a + b) / 2
9.         if f(c) > 0:
10.            b = c
11.        else:
12.            a = c
13.    return c
```

- Wypisz wartość **miejsce_zerowe(0.001)**.
- Sprawdź poprawność rozwiązania przez wyliczenie $f(x)$, gdzie x jest wynikiem otrzymanym po uruchomieniu programu.

Ćwiczenie 4. Funkcja malejąca w przedziale

Napisz program, za pomocą którego znajdziesz miejsce zerowe funkcji określonej wzorem $f(x) = -\frac{1}{4}x^2 + x + 2$ w przedziale $\langle 2, 10\rangle$ z dokładnością 0,001. Wykorzystaj algorytm połowienia i wyznacz miejsce zerowe, obliczając wyróżnik. Porównaj oba wyniki. Możesz też poszukać drugiego miejsca zerowego w innym przedziale.

PODSUMOWANIE

- Metoda połowienia umożliwia sprawne wyszukiwanie liczb w uporządkowanej liście liczb lub przedziale liczbowym dla wartości całkowitych i rzeczywistych. Jest znacznie szybsza od wyszukiwania liniowego.
- Specyfika liczb rzeczywistych i reprezentacja liczb zmiennoprzecinkowych zakładają określoną dokładność obliczeń. Trzeba o tym pamiętać podczas implementowania algorytmów na liczbach rzeczywistych, a zwłaszcza podczas porównywania dwóch liczb.
- Dzięki zastosowaniu metody połowienia można obliczyć wartość pierwiastka z danej liczby większej lub równej 1 oraz – dla pewnej grupy funkcji – miejsce zerowe. Obliczenia wykonuje się z zadanym przybliżeniem.
- Funkcja **abs()** przekazuje wartość bezwzględną podanej wartości. Jej argumentem może być liczba całkowita lub rzeczywista.

PYTANIA SPRAWDZAJĄCE

1. Jak są reprezentowane liczby zmiennoprzecinkowe w komputerze?
2. Jakie wartości początkowe można przyjąć dla obliczania pierwiastka, a jakie dla znajdowania miejsca zerowego?
3. Czy i jak podane przybliżenie wpływa na liczbę iteracji w algorytmie połowienia?

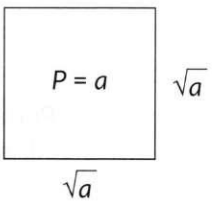
ZADANIE DODATKOWE

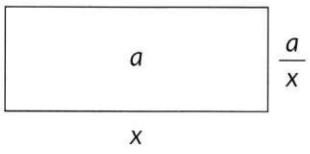
Zadanie 1. Metoda Newtona-Raphsona

Innym algorytmem obliczania pierwiastka jest metoda Newtona-Raphsona. Znajdź informacje o tej metodzie i jej zastosowaniach, a następnie zaimplementuj ją do obliczania pierwiastka kwadratowego z liczby i sprawdź, czy jest wydajniejsza od metody połowienia.

Wskazówka

Znajdowanie pierwiastka kwadratowego można sprowadzić do szukania długości boku kwadratu o zadanym polu.





Szukany jest $\sqrt{a}$, czyli bok kwadratu o polu a. Jeżeli zacząć od prostokąta o bokach x i a/x, gdzie za x można przyjąć np. 1, to obliczanie kolejnych przybliżeń x pozwoli znaleźć szukany pierwiastek.

$$x = \frac{1}{2}(x + \frac{a}{x})$$

Dokładność można zweryfikować przez wyznaczenie różnicy długości boków prostokąta $x - \frac{a}{x}$.

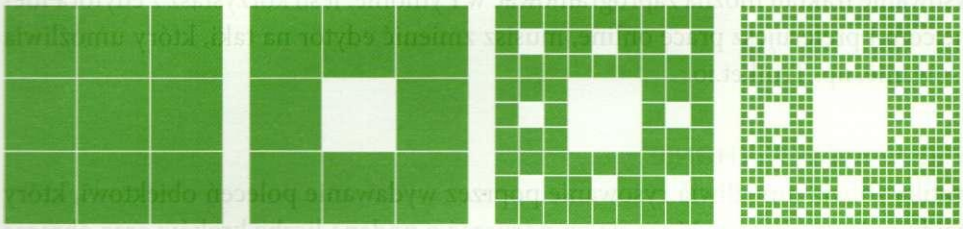
4. Fraktale

- Definiowanie fraktali
- Grafika żółwia
- Krzywa i płatek Kocha, drzewo binarne

ZADANIE NA START

Fraktal z kwadratów

Dywan Sierpińskiego powstaje poprzez podzielenie kwadratu na dziewięć mniejszych kwadratów i usunięcie środkowego kwadratu, a następnie ponowne zastosowanie tej samej procedury do każdego z pozostałych ośmiu kwadratów. W kolejnych krokach postępujemy tak jak poprzednio. Przeanalizuj poniższy rysunek. Ile pustych pól będzie po n-tym kroku?



krok $n = 0$ → zielony kwadrat

krok $n = 1$ → 1 puste pole

krok $n = 2$ → $1 + 8$ pustych pól

krok $n = 3$ → $1 + 8 + 8^2$ pustych pól

CO TO JEST FRAKTAL?

Niejedna osoba zachwyca się pięknem fraktali – obiektów otrzymywanych przez powtarzanie nieskończenie wiele razy tej samej operacji, w efekcie czego mały fragment obiektu ma taki sam kształt jak cały obiekt lub jego znaczna część. Te niezwykłe samopodobne obiekty znajdują zastosowanie w różnych dziedzinach nauki, techniki, ekonomii i sztuki, np. w medycynie wykorzystuje się je m.in. do analizy obrazów tomograficznych, w grafice i animacji komputerowej spotykanej w filmach często służą do generowania krajobrazów, z kolei w weterynarii wzory i charakterystyki fraktali pomagają w ocenie stresu cieplnego zwierząt hodowlanych, a w ekonomii ułatwiają przewidywanie zachowania notowań akcji.

POSZUKAJ W INTERNECIE

Dowiedz się, kto jest twórcą pojęcia *fraktal*. Sprawdź, jak wygląda fraktal, który ma w nazwie jego nazwisko.

Rys. 1. Kadr z dokumentu *Ukryty wymiar. Fraktale* w reżyserii Michaela Schwarza i Billa Jerseya, w którym omawia się generowanie fraktalnego krajobrazu w filmie *Star Trek II: Gniew Khana* (1982)

Jednym z fraktali jest zaprezentowany w zadaniu na start dywan Sierpińskiego. Inne słynne fraktale to krzywa Kocha, trójkąt Sierpińskiego, smok Heighwaya oraz zbiór Julii.

Rysowanie fraktali można zaprogramować w Pythonie. Jeśli korzystasz z edytora Ideone.com i preferujesz pracę online, musisz zmienić edytor na taki, który umożliwia rysowanie, np. Trinket.io.

RYSOWANIE W PYTHONIE

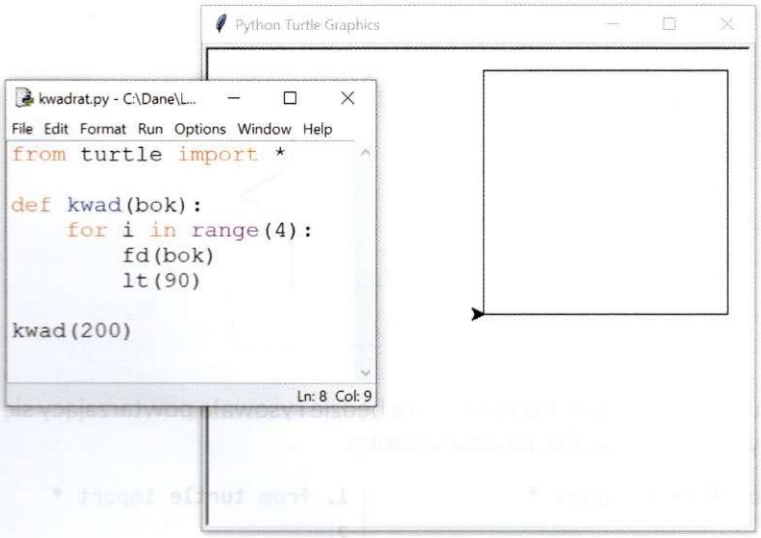
Grafika żółwia umożliwia rysowanie poprzez wydawanie poleceń obiektowi, który znajduje się na ekranie. Może się on poruszać o podaną liczbę kroków oraz obracać o zadany kąt. Stanowi podstawę języka Logo. Obecnie grafikę żółwia wykorzystuje się w różnych środowiskach programistycznych. W Pythonie odpowiada za nią moduł **`turtle`**, który trzeba importować za pomocą polecenia **`from turtle import *`**.

W poniższej tabeli zawarto podstawowe polecenia, które można wydawać żółwiowi.

Polecenie	Znaczenie	Opis działania
`fd(n)`	*forward* – naprzód	Żółw przesuwa się w aktualnym kierunku o *n* kroków.
`bk(n)`	*backward* – wstecz	Żółw przesuwa się wstecz o *n* kroków.
`rt(alfa)`	*right* – prawo	Żółw obraca się o podany kąt w stopniach w prawo.
`lt(alfa)`	*left* – lewo	Żółw obraca się o podany kąt w stopniach w lewo.
`pu()`	*pen up* – podnieś pisak	Żółw podnosi pisak – nie rysuje podczas przemieszczania się.
`pd()`	*pen down* – opuść pisak	Żółw opuszcza pisak – rysuje podczas przemieszczania się.

Żółw zawsze startuje ze środka ekranu z opuszczonym pisakiem i jest skierowany w prawą stronę. Jak napisać funkcję, po wywołaniu której zostanie narysowany kwadrat o boku podanym jako parametr? Najpierw trzeba zaimportować moduł **`turtle`**.

Potem należy zdefiniować funkcję, np. **kwad(bok)**, a w niej czterokrotnie powtórzyć rysowanie boku **fd(bok)** i obrót o 90° w lewo (lub prawo) **lt(90)** (lub **rt(90)**) – warto wykorzystać pętlę **for**. Teraz pozostaje wywołać funkcję z podaną wartością długości boku kwadratu i uruchomić program.



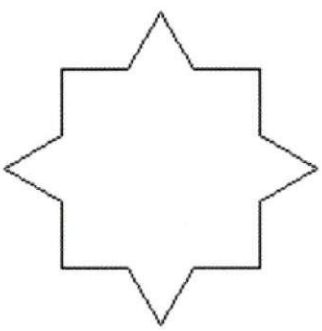
Rys. 2. Rysowanie kwadratu z wykorzystaniem modułu **turtle**

Aby kontrolować rysowanie – spowolnić je lub przyspieszyć – warto wykorzystać funkcję **speed()** i odpowiednio ustawić prędkość poruszania się żółwia:

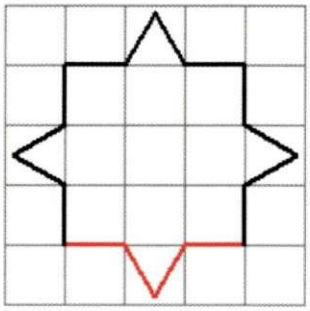
- 0 – najszybciej;
- 1 – najwolniej;
- 3 – wolno;
- 6 – normalnie;
- 10 – szybko.

Ćwiczenie 1. Gwiazda

Napisz funkcję **gwiazda(bok)**, po wywołaniu której powstanie motyw jak na rysunku poniżej. Parametr **bok** służy do określenia długości boku kwadratu.



- Przeanalizuj wzór – musisz cztery razy powtórzyć rysowanie fragmentu zaznaczonego kolorem czerwonym. Skoro **bok** określa długość boku kwadratu wynoszącą trzy kratki, to odcinek równy jednej kratce ma długość **bok/3**. Narysowanie ząbka wymaga skrętu w prawo o 60°, pokonania długości **bok/3**, skrętu w lewo o 120°, pokonania długości **bok/3** i skrętu w prawo o 60°.



- Zdefiniuj funkcję **fragment(bok)**, która będzie rysowała powtarzający się fragment gwiazdy. Możesz to zrobić na dwa sposoby.

```
1. from turtle import *
2.
3. def fragment(bok):
4.     fd(bok/3)
5.     rt(60)
6.     fd(bok/3)
7.     lt(120)
8.     fd(bok/3)
9.     rt(60)
10.    fd(bok/3)
```

```
1. from turtle import *
2.
3. def fragment(bok):
4.     fd(bok/3); rt(60)
5.     fd(bok/3); lt(120)
6.     fd(bok/3); rt(60)
7.     fd(bok/3)
```

- Zdefiniuj funkcję **gwiazda(bok)**, która z użyciem pętli **for** narysuje gwiazdę. Zauważ, że podobnie jak było w kwadracie, po narysowaniu jednego fragmentu żółw musi obrócić się o 90°.

```
1. def gwiazda(bok):
2.     for i in range(4):
3.         fragment(bok)
4.         lt(90)
```

- Przetestuj program. W razie potrzeby przyspiesz rysowanie za pomocą funkcji **speed(0)**.

```
1. speed(0)
2. gwiazda(100)
```

- Sprawdź działanie skryptu z różnymi danymi.

KRZYWA KOCHA

Szwedzki matematyk Niels Fabian Helge von Koch opisał krzywą fraktalną nazwaną jego imieniem w 1904 r. Krzywa ta powstaje z odcinka i ma nieskończoną długość.

- Krzywa Kocha stopnia 0. ma postać odcinka o długości a.

Rys. 3. Krzywa Kocha o stopniu złożoności $n = 0$

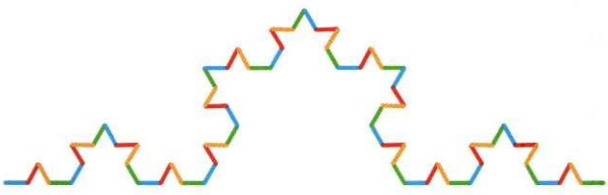
- Aby utworzyć krzywą stopnia 1., odcinek a trzeba podzielić na trzy części, przy czym środkowa część zyskuje kształt ząbka o ramieniu długości $a/3$, nachylonego do poziomu pod kątem 60°. Można powiedzieć, że krzywa stopnia 1. to cztery krzywe stopnia 0.

Rys. 4. Krzywa Kocha o stopniu złożoności $n = 1$

- Aby powstała krzywa stopnia 2., każdy z czterech odcinków krzywej stopnia 1. musi zostać podzielony na kolejne trzy części – według tej samej zasady co poprzednio. Każdy fragment (odcinek) zostaje zastąpiony krzywą stopnia 1. o długości trzy razy mniejszej niż poprzednie.

Rys. 5. Krzywa Kocha o stopniu złożoności $n = 2$

- Aby powstała krzywa stopnia 3., każda z czterech części musi zostać zastąpiona krzywą stopnia 2. o długości trzy razy mniejszej niż poprzednie – według tej samej zasady co poprzednio.



Rys. 6. Krzywa Kocha o stopniu złożoności $n = 3$

- I tak dalej.

W ten sposób krzywa stopnia 1. składa się z czterech odcinków, krzywa stopnia 2. – z 16 odcinków, krzywa stopnia 3. – z 64 odcinków. W dalszych stopniach – z 256 odcinków, 1024 odcinków, … .

Ćwiczenie 2. Krzywe Kocha dla konkretnych wartości

Napisz funkcję **koch0(bok)**, po wywołaniu której będzie rysowała się krzywa Kocha dla **n = 0**. Następnie napisz funkcje **koch1(bok)**, **koch2(bok)** i **koch3(bok)**, które będą korzystać z poprzednich funkcji, np. funkcja dla **n = 1** będzie korzystała z funkcji dla **n = 0**, a funkcja dla **n = 2** będzie korzystała z funkcji dla **n = 1**.

- Zaimportuj moduł **turtle**.
- Zdefiniuj funkcję **koch0(bok)**, odpowiedzialną za rysowanie krzywej dla parametru **0**.

```
1. from turtle import *
2.
3. def koch0(bok):
4.     fd(bok)
```

- Zdefiniuj funkcję **koch1(bok)**, odpowiedzialną za rysowanie krzywej dla parametru **1** – wykorzystaj cztery wywołania **koch0(bok/3)**.

```
1. def koch1(bok):
2.     koch0(bok/3)
3.     lt(60)
4.     koch0(bok/3)
5.     rt(120)
6.     koch0(bok/3)
7.     lt(60)
8.     koch0(bok/3)
```

- Zdefiniuj funkcję **koch2(bok)**, odpowiedzialną za rysowanie krzywej dla parametru **2**. Każdy prosty odcinek poprzedniej krzywej zastąp wywołaniem funkcji **koch1(bok/3)**.

```
1. def koch2(bok):
2.     koch1(bok/3)
3.     lt(60)
4.     koch1(bok/3)
5.     rt(120)
6.     koch1(bok/3)
7.     lt(60)
8.     koch1(bok/3)
```

- Przetestuj kolejne funkcje. Sprawdź poprawność rysunków.

- Zdefiniuj funkcję **koch3(bok)**, odpowiedzialną za rysowanie krzywej dla parametru **3**.

```
1. def koch3(bok):
2.     pass
```

- Zdefiniuj kolejne funkcje.

W ten sposób można pisać funkcje, które będą wywoływać poprzednią funkcję dla parametru **n - 1**. Aby zapisać ogólną postać rysowania krzywej Kocha stopnia **n**, trzeba skorzystać z tej samej funkcji wywoływanej z parametrem **n - 1**. Rysowanie krzywej zakończy się dla parametru **0** – wystarczy podać polecenie **return** bez parametru, a funkcja się zakończy.

ZAPAMIĘTAJ!

Sposób definiowania funkcji, który polega na umieszczeniu w treści funkcji odwołań do niej samej, nazywa się rekurencją.

Ćwiczenie 3. Krzywa Kocha z podanym stopniem

Napisz funkcję **koch(bok, n)**, po wywołaniu której krzywe będą rysowane zgodnie z wartością podanych parametrów. Wzoruj się na funkcjach pisanych w ćwiczeniu 2.

- Zaimportuj moduł **turtle**.
- Po deklaracji funkcji **koch(bok, n)** napisz warunek zatrzymania funkcji. Dla najmniejszego parametru **n = 0** żółw będzie wówczas rysował odcinek i kończył działanie funkcji.
- Dla czterech odcinków krzywej Kocha wywołaj funkcję **koch(bok/3, n - 1)**. Kąty obrotów będą identyczne jak w poprzednim ćwiczeniu.

```
1. from turtle import *
2.
3. def koch(bok, n):
4.     if n == 0:
5.         fd(bok)
6.         return
7.     koch(bok/3, n - 1)
8.     lt(60)
9.     koch(bok/3, n - 1)
10.    rt(120)
11.    koch(bok/3, n - 1)
12.    lt(60)
13.    koch(bok/3, n - 1)
```

- Przetestuj rozwiązanie. Wywołaj funkcję **koch(bok, n)** dla boku wielkości **300**

i stopni **1**, **2**, **3**, **4** i **5**. Sprawdź, czy kolejne krzywe są rysowane poprawnie. Zauważ, że rysunki dla kolejnych wartości parametru **n** stają się coraz mniej czytelne, a rysowanie jest coraz dłuższe. Wynika to z liczby odcinków – skoro dla **n = 5** jest ich 1024, to dla **n = 6** jest ich już 4096.

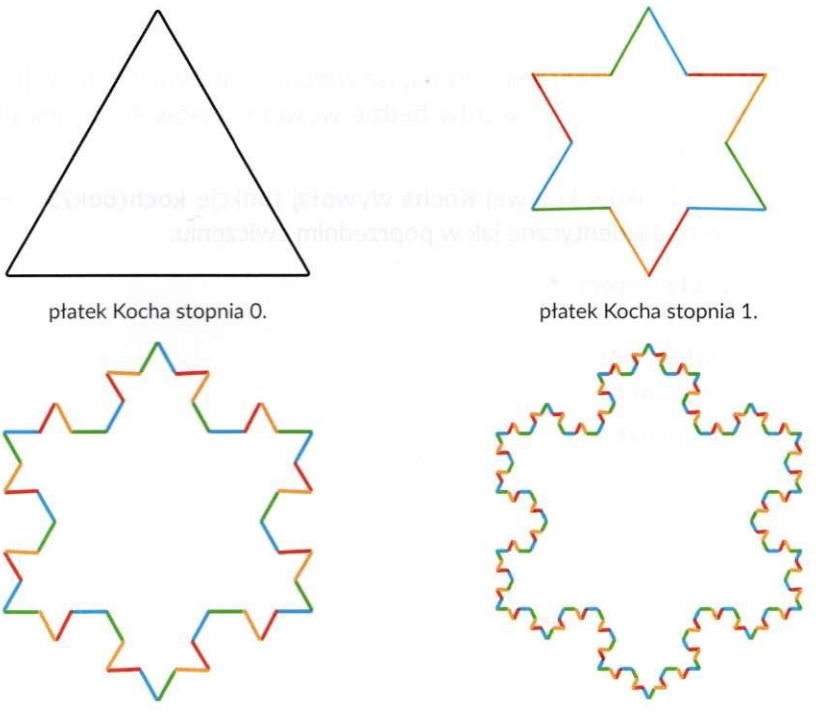
WIEM WIĘCEJ

Po wybraniu opcji **Turtle Demo** w Pomocy edytora IDLE otworzy się okno **Python turtle-graphics examples**. Można w nim uruchomić i obejrzeć ciekawe animacje, np. animacja **fractalCurves** prezentuje wypełniającą płaszczyznę krzywą Hilberta oraz ciekawą kombinację krzywych Kocha.

PŁATEK KOCHA

Płatek Kocha jest trójkątem równobocznym, którego brzeg stanowi krzywa Kocha. Dla stopnia złożoności 0 rysuje się trójkąt równoboczny o zadanym boku. Podczas rysowania płatka o złożoności 1 każdy bok jest zastępowany krzywą Kocha stopnia 1. Podczas rysowania płatka o złożoności 2 każdy bok jest zastępowany krzywą Kocha stopnia 2. Podczas rysowania płatka o złożoności *n* każdy bok jest zastępowany krzywą Kocha stopnia *n*.

Na rysunkach poniżej widać trójkąt równoboczny i krzywe Kocha zastępujące jego boki.



Rys. 7. Kolejne płatki Kocha

Ćwiczenie 4. Tworzenie płatka Kocha

Napisz funkcję **platek(n)**, po wywołaniu której zostanie narysowany płatek Kocha stopnia **n**.

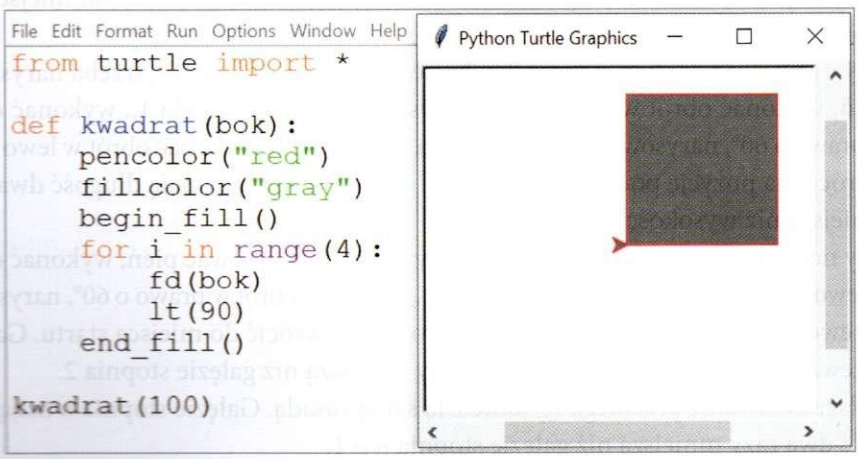
- Wykorzystaj skrypt z ćwiczenia 3.
- Nadaj wartość zmiennej **bok**.
- Obróć żółwia w lewo o **60°**, aby narysować trójkąt równoboczny.
- Wykorzystaj pętlę **for** do narysowania trójkąta równobocznego, którego bok będzie zastąpiony krzywą Kocha stopnia **n** podanego jako parametr. Po narysowaniu boku obróć żółwia o **120°**.
- Po zakończeniu rysowania obróć żółwia w prawo o **60°**, aby wrócił do pozycji początkowej.

```
1. def platek(n):
2.     bok = 300
3.     lt(60)
4.     for i in range(3):
5.         koch(bok, n)
6.         rt(120)
7.     rt(60)
```

- Przetestuj rozwiązanie.

Rysunki można rysować kolorowym pisakiem oraz zamalowywać ich wnętrze. Należy wówczas ustalić kolor pisaka i wypełnienia. Służą do tego następujące polecenia:

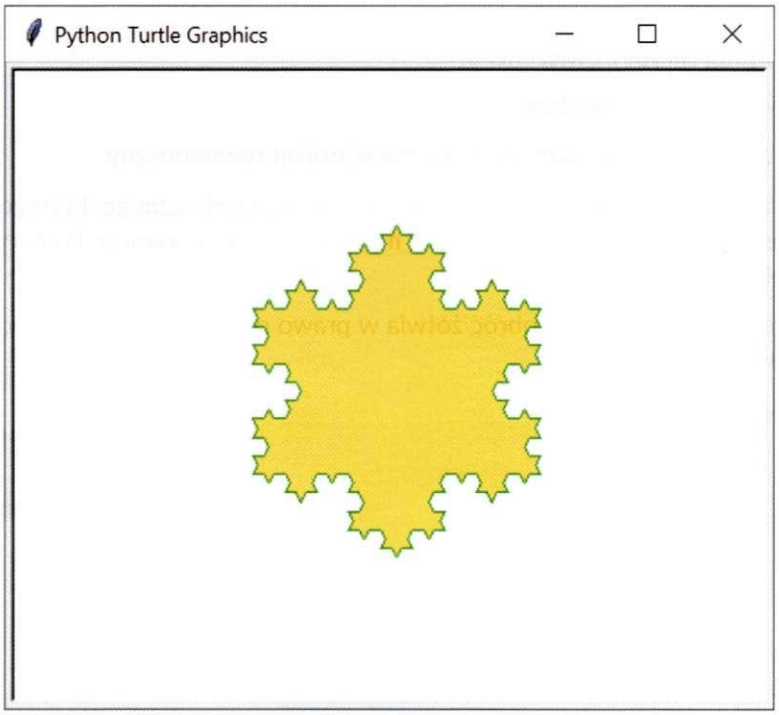
- **pencolor("green")** – ustala kolor pisaka na zielony;
- **fillcolor("blue")** – ustala kolor wypełnienia na niebieski;
- **begin_fill()** – początek wypełniania figury;
- **end_fill()** – koniec wypełniania figury.



Rys. 8. Przykładowy program realizujący rysowanie kwadratu o czerwonej krawędzi i szarym wypełnieniu

Ćwiczenie 5. Kolorowy płatek Kocha

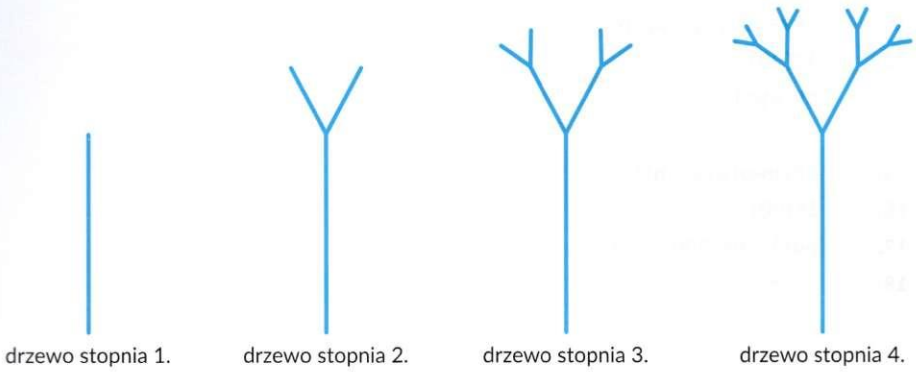
Uzupełnij kod funkcji **`platek(n)`** tak, aby żółw rysował wybranymi kolorami.



DRZEWO BINARNE

Fraktalne drzewa binarne to takie, w których z pnia wyrastają dwie gałęzie, a następnie z każdej gałęzi wyrastają dwie kolejne gałęzie i tak dalej. Kąt między gałęziami może mieć dowolną wartość, także losową.

- Aby narysować drzewo stopnia 1., należy narysować pień i wrócić do miejsca, od którego rozpoczęło się rysowanie.
- Aby narysować drzewo stopnia 2. o kącie między gałęziami 60°, trzeba narysować pień, wykonać obrót w lewo o 30°, narysować drzewo stopnia 1., wykonać obrót w prawo o 60°, narysować drzewo stopnia 1., ponownie wykonać obrót w lewo o 30° i wrócić na pozycję początkową. Gałęzie drzewa stopnia 2. mają długość dwa razy mniejszą niż wysokość pnia.
- Aby narysować drzewo stopnia 3., należy kolejno: narysować pień, wykonać obrót w lewo o 30°, narysować drzewo stopnia 2., wykonać obrót w prawo o 60°, narysować drzewo stopnia 2., wykonać obrót w lewo o 30° i wrócić do miejsca startu. Gałęzie drzewa stopnia 3. mają długość dwa razy mniejszą niż gałęzie stopnia 2.
- Dalsze rysowanie przebiega zgodnie z tą samą zasadą. Gałęzie stopnia n mają długość dwa razy mniejszą niż gałęzie stopnia $n - 1$.



Rys. 9. Kolejne drzewa binarne

Ćwiczenie 6. Fraktalne drzewo binarne

Napisz funkcję **drzewo(wys, n)**, po wywołaniu której zostanie narysowane drzewo binarne stopnia **n** o wysokości pnia równej **wys**. Kąt odchylenia gałęzi ustal na 60°.

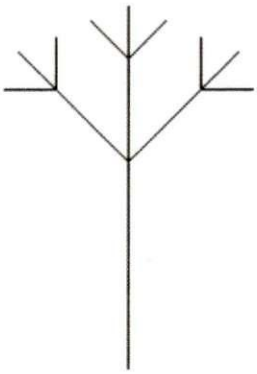
- Zaimportuj moduł **turtle**.
- Zacznij od funkcji pomocniczej **drzewo1(wys, n)**, po wywołaniu której żółw będzie rysował drzewo.
- Po deklaracji funkcji pomocniczej **drzewo1(wys, n)** napisz warunek zatrzymania funkcji. Dla najmniejszego parametru **n = 1** żółw narysuje wówczas odcinek długości podanej jako parametr, wróci do punktu wyjścia i zakończy działanie funkcji.
- Uzupełnij funkcję pomocniczą **drzewo1(wys, n)** o rysowanie drzewa stopnia **n** – narysowanie pnia, obrót w lewo o **30°**, narysowanie drzewa stopnia **n - 1**, obrót w prawo o **60°**, narysowanie drzewa stopnia **n - 1**, obrót w lewo o **30°** i powrót do punktu wyjścia. Gałęzie drzewa stopnia **n - 1** powinny mieć **wys/2**.
- Napisz funkcję główną **drzewo(wys, n)**, która odpowiada za obracanie żółwia tak, aby stał skierowany pionowo do góry, oraz za przesuwanie go w dół, aby rysunek był dobrze widoczny na całym obszarze okna graficznego, i zawiera wywołanie funkcji pomocniczej **drzewo1(wys, n)**.

```
1. from turtle import *
2.
3. def drzewo1(wys, n):
4.     if n == 1:
5.         fd(wys); bk(wys)
6.         return
7.     fd(wys)
8.     lt(30)
9.     drzewo1(wys/2, n - 1)
10.    rt(60)
```

```
11.     drzewo1(wys/2, n - 1)
12.     lt(30)
13.     bk(wys)
14.
15. def drzewo(wys, n):
16.     lt(90)
17.     pu(); bk(200); pd()
18.     drzewo1(wys, n)
```

Ćwiczenie 7. Drzewo z trzema gałęziami

Zmodyfikuj funkcję **drzewo1(wys, n)** z ćwiczenia 6 tak, aby w wyniku dla parametru **n = 3** zostało narysowane drzewo takie jak na rysunku poniżej. Kąt odchylenia gałęzi od pionu wynosi **45°**. Gałęzie drzewa stopnia **n = 1** są dwa razy krótsze od pnia. Każde kolejne gałęzie są dwa razy krótsze od poprzednich.



WIEM WIĘCEJ

System Lindenmayera, zwany w skrócie L-systemem, to zestaw reguł służący do generowania graficznych struktur o budowie fraktalnej. Węgierski biolog Aristid Lindenmayer stworzył go w 1968 r. w celu modelowania za jego pomocą zachowania komórek roślin. Każdy symbol w L-systemie jest w takim modelu interpretowany jako określona sekwencja ruchów żółwia.

PODSUMOWANIE

- Cechą charakterystyczną fraktala jest to, że jego fragment jest podobny do całości. Fraktale powstają przez wielokrotne powtarzanie prostych operacji.
- Wykorzystując moduł **turtle** Pythona, można korzystać z poleceń grafiki żółwia do rysowania na ekranie graficznym.

- Krzywa Kocha stopnia 0. to po prostu odcinek. Podczas rysowania krzywej Kocha stopnia 1. dzieli się poprzedni odcinek na trzy części i środkową z nich zastępuje dwoma odcinkami tej samej długości co pozostałe dwa odcinki. W identyczny sposób tworzy się krzywe Kocha kolejnych stopni.
- Płatek Kocha jest trójkątem równobocznym, którego brzeg stanowi krzywa Kocha.
- Sposób definiowania funkcji, który polega na umieszczeniu w treści funkcji odwołań do niej samej, nosi miano rekurencji.
- Drzewa binarne to takie, w których z pnia wyrastają dwie gałęzie, a następnie z każdej gałęzi wyrastają dwie kolejne gałęzie.

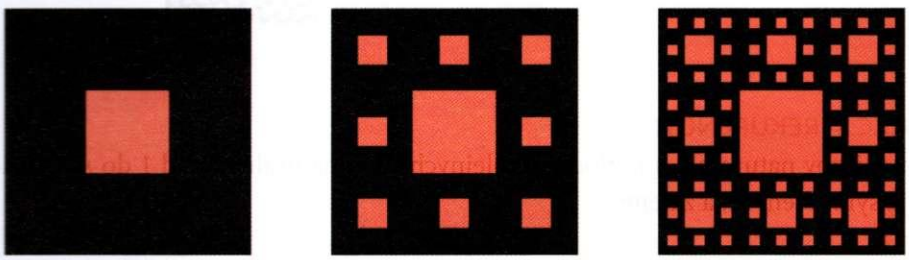
PYTANIA SPRAWDZAJĄCE

1. Jaki moduł odpowiada za rysowanie z wykorzystaniem grafiki żółwia?
2. Z czego składa się płatek Kocha?
3. Jaka jest zasada rysowania fraktalnego drzewa binarnego?

ZADANIA DODATKOWE

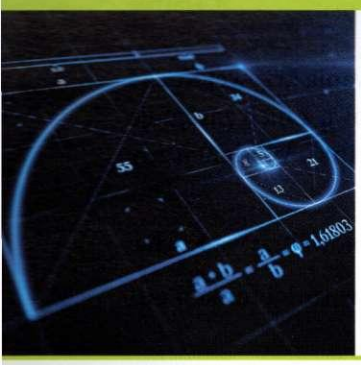
Zadanie 1. Dywan Sierpińskiego

Napisz funkcję **dywan(bok, n)**, po wywołaniu której na ekranie zostanie narysowany dywan Sierpińskiego o różnym stopniu złożoności.



Zadanie 2. Trójkąt Sierpińskiego

Napisz funkcję rysowania trójkąta Sierpińskiego o różnej złożoności. Przeanalizuj zaprezentowaną poniżej zasadę konstrukcji tego obiektu.



5. Rekurencja i ciąg Fibonacciego

- Definiowanie funkcji rekurencyjnych
- Iteracja i rekurencja
- Zalety i wady rekurencji

ZADANIE NA START

Spirala Fibonacciego

Przyjrzyj się uważnie poniższemu rysunkowi. Do kwadratu o boku 1 przylega kwadrat o takim samym boku. Kolejny bok jest już większy: 1 + 1 = 2. A kolejny? Wypisz długości boków coraz większych kwadratów. Czy widzisz jakieś zależności między nimi?

FUNKCJA REKURENCYJNA

Silnia liczby naturalnej n to iloczyn kolejnych liczb naturalnych od 1 do n. Oznacza się ją symbolem $n!$, a zatem:

$0! = 1$

$1! = 1$, czyli $0! \cdot 1$

$2! = 1 \cdot 2$, czyli $1! \cdot 2$

$3! = 1 \cdot 2 \cdot 3$, czyli $2! \cdot 3$

$4! = 1 \cdot 2 \cdot 3 \cdot 4$, czyli $3! \cdot 4$

...

$n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \ldots \cdot (n-1) \cdot n$, czyli $(n-1)! \cdot n$

Można zdefiniować silnię jako funkcję, której wartość jest równa iloczynowi $(n - 1)!$ i n, z założeniem, że $0! = 1$. Bardziej formalnie można to zapisać następująco:

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

Należy zauważyć, że w zapisie obliczania $n!$ jest odwołanie do obliczania $(n-1)!$, czyli tej samej funkcji z innym parametrem. Aby wyznaczyć $n!$, trzeba pomnożyć wynik $(n-1)!$ przez n dla $n > 0$, a dla samego $n = 0$ to po prostu 1.

Takie wywołanie przez funkcję samej siebie – czy też ogólniej: rozwiązanie bardziej złożonego problemu na podstawie rozwiązania mniejszego problemu – nosi miano **rekurencji**. Trzeba zaznaczyć, że definicja rekurencyjna potrzebuje nierekurencyjnego przypadku bazowego, który zagwarantuje zakończenie jej wykonywania. W przypadku funkcji obliczania silni warunkiem zatrzymania jest wartość dla $n = 0$ wynosząca 1. Kolejne kroki obliczeń, czyli mnożenie wartości $(n-1)!$ przez n, to nic innego jak wywołanie rekurencyjne tej samej funkcji dla mniejszego parametru.

Dlaczego warunek zatrzymania jest tak ważny?

1. Oblicz 4! jako iloczyn 3! i 4.
2. Oblicz 3! jako iloczyn 2! i 3.
3. Oblicz 2! jako iloczyn 1! i 2.
4. Oblicz 1! jako iloczyn 0! i 1.
   - Tę wartość możesz już wyznaczyć, bo z definicji $0! = 1$ (jest to warunek zatrzymania). Masz pierwszy wynik: $1! = 0! \cdot 1 = 1$.
   - Gdyby w tym punkcie nie było warunku zatrzymania, prowadziłoby to do nieskończonych obliczeń.
5. Jeśli wiesz, że $1! = 1$, oblicz $2! = 1! \cdot 2 = 1 \cdot 2 = 2$.
6. Jeśli wiesz, że $2! = 2$, oblicz $3! = 2! \cdot 3 = 2 \cdot 3 = 6$.
7. Jeśli wiesz, że $3! = 6$, oblicz $4! = 3! \cdot 4 = 6 \cdot 4 = 24$.

ZAPAMIĘTAJ!

Rozwiązanie problemu za pomocą rekurencji polega na rozkładzie na mniejsze problemy, z których można zbudować rozwiązanie problemu wyjściowego i które prowadzą w końcu do problemów z rozwiązaniem bezpośrednim.

Ćwiczenie 1. Silnia rekurencyjnie

Przeanalizuj dane w tabeli i zdefiniuj funkcję rekurencyjną **silnia(n)**, której argumentem jest liczba nieujemna **n**, a wynikiem – obliczona silnia podanej liczby **n**.

Wywołanie funkcji	Wynik
silnia(3)	**6**
silnia(9)	**362880**

- Po deklaracji funkcji **silnia(n)** napisz warunek zatrzymania dla **n = 0**. Wynikiem funkcji jest wtedy **1**.

- Zapisz iloczyn wywołania rekurencyjnego dla problemu **n - 1** i **n**. Słowo kluczowe **return** kończy działanie funkcji i przekazuje wartość.

```
1. def silnia(n):
2.     if n == 0:
3.         return 1
4.     return silnia(n - 1) * n
```

- Sprawdź działanie funkcji dla różnych danych.

Aby się przekonać, jak szybko rośnie wielkość wyniku funkcji **silnia(n)**, można wykorzystać pętlę **for** w celu sprawdzania wyniku funkcji dla wielu wartości.

```
File  Edit  Format  Run  Options  Window  Help
def silnia(n):
    if n == 0:
        return 1
    return silnia(n - 1) * n

for i in range(51):
    print(i, silnia(i))
```

```
========== RESTART:
0 1
1 1
2 2
3 6
4 24
5 120
6 720
7 5040
8 40320
9 362880
10 3628800
11 39916800
12 479001600
13 6227020800
14 87178291200
15 1307674368000
16 20922789888000
17 355687428096000
18 6402373705728000
19 121645100408832000
20 2432902008176640000
21 51090942171709440000
22 1124000727777607680000
23 25852016738884976640000
24 620448401733239439360000
25 15511210043330985984000000
26 403291461126605635584000000
27 10888869450418352160768000000
28 304888344611713860501504000000
29 8841761993739701954543616000000
30 265252859812191058636308480000000
31 8222838654177922817725562880000000
32 263130836933693530167218012160000000
33 8683317618811886495518194401280000000
34 295232799039604140847618609643520000000
35 10333147966386144929666651337523200000000
36 371993326789901217467999448150835200000000
37 13763753091226345046315979581580902400000000
38 523022617466601111760007224100074291200000000
39 20397882081197443358640281739902897356800000000
40 815915283247897734345611269596115894272000000000
41 33452526613163807108170062053440751665152000000000
42 1405006117752879898543142606244511569936384000000000
43 60415263063373835637355132068513997507264512000000000
44 2658271574788448768043625811014615890319638528000000000
45 119622220865480194561963161495657715064383733760000000000
46 5502622159812088949850305428800254892961651752960000000000
47 258623241511168180642964355153611979969197632389120000000000
48 12413915592536072670862289047373375038521486354677760000000000
49 608281864034267560872252163321295376887552831379210240000000000
50 30414093201713378043612608166064768844377641568960512000000000000
```

Rys. 1. Szybki wzrost wartości funkcji **silnia(n)**

WIEM WIĘCEJ

W praktyce liczba wywołań rekurencyjnych funkcji jest ograniczona, dlatego dla dużych wartości parametru można spotkać się z komunikatem *RecursionError: maximum recursion depth exceeded in comparison.*

REKURENCJA OGONOWA I ITERACJA

Realizowana do tej pory rekurencja jest nazywana rekurencją ogonową, ponieważ rekurencyjne wywołanie funkcji znajduje się na końcu tej funkcji. Taką rekurencję można łatwo zamienić na iterację dzięki wykorzystaniu odpowiedniej pętli.

Ćwiczenie 2. Silnia iteracyjnie

Przeanalizuj dane w tabeli i zdefiniuj funkcję iteracyjną **silnia2(n)**, której argumentem jest liczba nieujemna **n**, a wynikiem – obliczona silnia podanej liczby **n**.

Wywołanie funkcji	Wynik
silnia2(3)	**6**
silnia2(9)	**362880**

- Przeanalizuj problem.
- Utwórz zmienną **iloczyn** i nadaj jej wartość **1** (wartość neutralna dla mnożenia).
- W pętli **for** obliczaj iloczyn kolejnych liczb: **1 · 2 · 3 · ... · n**.
- Wynikiem funkcji powinien być wyznaczony iloczyn.

```
1. def silnia2(n):
2.     iloczyn = 1
3.     for i in range(1, n + 1):
4.         iloczyn *= i
5.     return iloczyn
```

- Sprawdź działanie skryptu z różnymi danymi – dla wartości z tabeli i np. dla **n = 50**. Zwróć uwagę, że zapisanie wartości silni większych liczb, np. powyżej 1000, zajmuje kilkadziesiąt wierszy.

POTĘGOWANIE REKURENCYJNE

Kolejną funkcją matematyczną, której wzór można zapisać w sposób rekurencyjny, jest potęga stopnia n liczby a, równa iloczynowi a^{n-1} i a, z założeniem, że $a^0 = 1$.
Wzór przedstawia się następująco:

$$a^n = \begin{cases} 1 \text{ dla } n = 0 \\ a^{n-1} \cdot a \text{ dla } n > 0 \end{cases}$$

Jak obliczyć wartość 2^4?

1. Oblicz 2^4 jako iloczyn 2^3 i 2.
2. Oblicz 2^3 jako iloczyn 2^2 i 2.
3. Oblicz 2^2 jako iloczyn 2^1 i 2.
4. Oblicz 2^1 jako iloczyn 2^0 i 2.
5. Oblicz 2^0.
   - Tę wartość możesz już wyznaczyć, bo z definicji $2^0 = 1$ (jest to warunek zatrzymania). Masz pierwszy wynik.
6. Jeśli wiesz, że $2^0 = 1$, oblicz $2^1 = 2^0 \cdot 2 = 2$.
7. Jeśli wiesz, że $2^1 = 2$, oblicz $2^2 = 2^1 \cdot 2 = 2 \cdot 2 = 4$.
8. Jeśli wiesz, że $2^2 = 4$, oblicz $2^3 = 2^2 \cdot 2 = 4 \cdot 2 = 8$.
9. Jeśli wiesz, że $2^3 = 8$, oblicz $2^4 = 2^3 \cdot 2 = 8 \cdot 2 = 16$.

Ćwiczenie 3. Potęga rekurencyjnie

Przeanalizuj dane w tabeli i zdefiniuj funkcję rekurencyjną **potega(a, n)**, której argumentami są liczba **a** oraz liczba **n**, a wynikiem – obliczona **n**-ta potęga podanej liczby **a**.

Wywołanie funkcji	Wynik
`potega(2, 3)`	`8`
`potega(9, 5)`	`59049`

- Po deklaracji funkcji **potega(a, n)** napisz warunek zatrzymania dla `n = 0`. Wynikiem funkcji jest wtedy `1`.
- Zapisz iloczyn wywołania rekurencyjnego dla problemu `n - 1` i **a**. Słowo kluczowe `return` kończy działanie funkcji i przekazuje wartość.

```
1. def potega(a, n):
2.     if n == 0:
3.         return 1
4.     return potega(a, n - 1) * a
```

- Sprawdź działanie skryptu z różnymi danymi.

Ćwiczenie 4. Rekurencyjny ciąg liczb

Pewien ciąg liczb został zdefiniowany rekurencyjnie, tak jak poniżej:

$$a_n = \begin{cases} 1 \text{ dla } n = 0 \\ -n \cdot a^{n-1} + 4 \text{ dla } n > 1 \end{cases}$$

Zdefiniuj funkcję rekurencyjną **ciag(n)** obliczania **n**-tego wyrazu tego ciągu.

Wywołanie funkcji	Wynik
ciag(5)	**-56**
ciag(8)	**19012**

Ćwiczenie 5. Suma cyfr

Przeanalizuj dane w tabeli i zdefiniuj funkcję rekurencyjną **suma_cyfr(n)**, która wyznaczy sumę cyfr liczby podanej jako parametr.

Wywołanie funkcji	Wynik
suma_cyfr(136)	**10**
suma_cyfr(19918)	**28**

- Przeanalizuj problem. Zapisz definicję rekurencyjną sumy cyfr podanej liczby.

$$\text{suma}(n) = \begin{cases} 0 \text{ dla } n = 0 \\ \text{suma(bez ostatniej cyfry)} + n \,\%\, 10 \text{ dla } n > 0) \end{cases}$$

- Po deklaracji funkcji **suma_cyfr(n)** napisz warunek zatrzymania dla **n = 0**. Wynikiem funkcji jest wtedy **0**.
- Zapisz sumę wywołania rekurencyjnego dla problemu **n - 1** i wartości ostatniej cyfry. Problem **n - 1** to suma cyfr liczby **n** bez ostatniej cyfry, czyli **n // 10**. Ostatnia cyfra to **n % 10**.

```
1. def suma_cyfr(n):
2.     if n == 0:
3.         return 0
4.     return suma_cyfr(n // 10) + n % 10
```

- Sprawdź działanie skryptu z różnymi danymi.

CIĄG FIBONACCIEGO

Chyba najbardziej znanym i interesującym spośród wszystkich ciągów liczbowych jest ciąg Fibonacciego, który zawdzięcza swoją nazwę matematykowi z Pizy, Leonardowi Fibonacciemu. W 1202 r. wydał on księgę matematyczną *Liber Abaci*, w której opisał m.in. pozycyjny system liczbowy.

Ciąg Fibonacciego tworzą liczby naturalne o takiej własności, że dwa pierwsze wyrazy są równe 1, a każdy kolejny wyraz jest sumą dwóch poprzednich. Określa się go następująco:

$$F(n) = \begin{cases} 1 \text{ dla } n = 1 \text{ i } n = 2 \\ F(n-1) + F(n-2) \text{ dla } n > 2 \end{cases}$$

Początkowe wartości tego ciągu to: 1, 1, 2, 3, 5... .

Ćwiczenie 6. Elementy ciągu Fibonacciego

Oblicz pierwsze 10 wartości liczb ciągu Fibonacciego. Porównaj liczby uzyskane jako wynik w zadaniu na start z obliczonymi wartościami ciągu.

WIEM WIĘCEJ

W przyrodzie można znaleźć rośliny, których budowa lub właściwości odnoszą się do liczb Fibonacciego. Botanicy opisują drzewa, na których z każdej gałęzi w każdym następnym roku wyrasta jedna młoda gałązka. Ciąg Fibonacciego można również zauważyć na spodzie szyszki lub w układzie nasion słonecznika w kwiatostanie.

Ćwiczenie 7. Ciąg Fibonacciego rekurencyjnie

Przeanalizuj dane w tabeli i zdefiniuj funkcję rekurencyjną **fib(n)**, której argumentem jest liczba **n**, a wynikiem – obliczona **n**-ta liczba ciągu Fibonacciego.

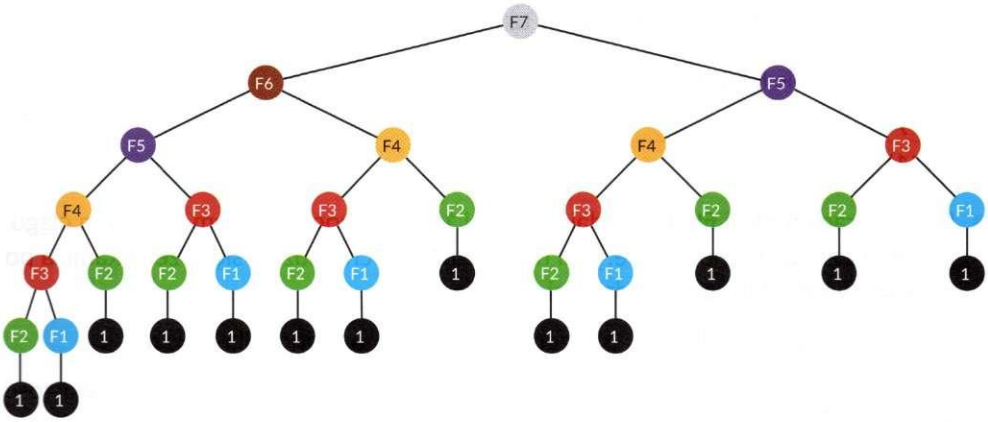
Wywołanie funkcji	Wynik
fib(4)	**3**
fib(11)	**89**

- Przeanalizuj problem. Rozpatrz przypadek, gdy **n** jest równe **1** lub **2**. Dla **n** równego **1** lub **2** wynikiem funkcji powinna być liczba **1**.
- Po deklaracji funkcji **fib(n)** zapisz warunek dla **n** równego **1** lub **2**, a następnie wywołanie rekurencyjne jako sumę funkcji **fib(n - 1)** i **fib(n - 2)**.

```
1. def fib(n):
2.     if n < 3:
3.         return 1
4.     return fib(n - 1) + fib(n - 2)
```

- Sprawdź działanie skryptu z różnymi danymi, m.in. wartościami 32, 37 i 40. Obserwuj szybkość pojawiania się wyniku na ekranie: dla **n = 32** wynik (2 178 309) pojawia się od razu, dla **n = 37** (wynik 24 157 817) obliczenia trwają dłużej, ale poniżej 10 s (wartości różnią się w zależności od klasy sprzętu), dla **n = 40** (wynik 102 334 155) obliczenia zabierają już kilkadziesiąt sekund.

Dlaczego – mimo niewielkich różnic w wartości parametru **n** – znacznie dłużej czeka się na wynik?



Rys. 2. Obliczanie siódmej liczby Fibonacciego – dla $n = 7$ wartość funkcji F5 jest obliczana dwa razy, F4 trzy razy, F3 pięć razy, ...

Liczba ciągu Fibonacciego (od trzeciej) jest obliczana jako suma dwóch poprzednich takich liczb. Jeśli stosujemy rekurencję, to funkcja dla tych samych wartości jest wywoływana wielokrotnie.

$$f(7) = f(6) + f(5)$$

$$f(7) = f(5) + f(4) + f(4) + f(3)$$

$$f(7) = f(4) + f(3) + f(3) + f(2) + f(3) + f(2) + f(2) + f(1)$$

$$f(7) = f(3) + f(2) + f(2) + f(1) + f(2) + f(1) + f(2) + f(2) + f(1) + f(2) + f(2) + f(1)$$

$$f(7) = f(2) + f(1) + f(2) + f(2) + f(1) + f(2) + f(1) + f(2) + f(2) + f(1) + f(2) + f(2) + f(1) = 13$$

Rys. 3. Rozwinięcie rekurencyjne $f(7)$ – na kolorowo zaznaczono liczby ciągu Fibonacciego, które dalej rozkładają się rekurencyjnie

W obliczeniach siódmej liczby ciągu Fibonacciego występuje pięć wywołań rekurencyjnych dla $n = 3$. Funkcja dla każdego z nich wylicza wartość niezależnie, nie korzysta z raz obliczonego wyniku. Dlatego dla coraz większych wartości parametrów obliczenia trwają coraz dłużej. Nie jest to korzystne. Zapis rekurencyjny generuje nadmiarowe obliczenia. Jest to jedna z wad rekurencji. Problem można rozwiązać przez zastosowanie podejścia bardziej ekonomicznego: od najmniejszych wartości do coraz większych. Nie będzie wtedy dublowania obliczeń, dzięki czemu wzrośnie szybkość działania funkcji.

POSZUKAJ W INTERNECIE

Znajdź informacje dotyczące udziału liczb ciągu Fibonacciego w przyrodzie, muzyce i literaturze.

Ćwiczenie 8. Ciąg Fibonacciego iteracyjnie

Przeanalizuj dane w tabeli i zdefiniuj funkcję iteracyjną **fib2(n)**, której argumentem jest liczba **n**, a wynikiem – obliczona **n**-ta liczba ciągu Fibonacciego.

Wywołanie funkcji	Wynik
fib2(4)	**3**
fib2(11)	**89**

- Po deklaracji funkcji **fib2(n)** nadaj wartości **1** dwóm początkowym wyrazom ciągu. Można to zrobić jednocześnie, jeśli nazwy zmiennych oddzieli się przecinkami, a po znaku równości tak samo oddzieli się ich wartości.
- Pętlę **for** rozpocznij dla **i = 3** (trzeci wyraz ciągu), a skończ dla **i = n**.
- Zmieniaj wartości zmiennych. Przy obliczaniu trzeciego wyrazu ciągu jego wartość to suma dwóch poprzednich (**a + b**).

```
1. def fib2(n):
2.     a, b = 1, 1
3.     for i in range(3, n + 1):
4.         a, b = b, a + b
5.     return b
```

- Sprawdź działanie skryptu z różnymi danymi. Zauważ, że zarówno dla **n = 32**, jak i dla **n = 37** oraz dla **n = 40** wynik jest natychmiastowy. Algorytm iteracyjny działa szybko nawet dla **n = 100**, **200**, **300**, ...

WIEM WIĘCEJ

Matematyka wiele zawdzięcza Leonardowi Fibonacciemu. Ciekawe informacje znajdziesz w filmie *Tajemniczy ciąg Fibonacciego. Złota liczba. Boska proporcja*, dostępnym m.in. na YouTubie na kanale *Pasja informatyki*: **https://www.youtube.com/watch?v=wb7kPaM8cfg**.

ZALETY I WADY REKURENCJI

Jak widać, rekurencja oprócz zalet ma jednak swoje słabe strony. Choć niewątpliwie wpływa na zwięzłość zapisu funkcji i niejednokrotnie pozwala na rozwiązanie problemów w sposób łatwiejszy niż iteracja, bywa nieefektywna czasowo (zdarza się, że jest wolniejsza od iteracji) i pamięciowo (podczas obliczeń jest zużywana spora ilość pamięci, ponieważ każde wywołanie funkcji wymaga zapamiętania informacji, które są niezbędne do odtworzenia stanu sprzed wywołania, co łatwo można zaobserwować podczas próby wyznaczenia setnego wyrazu ciągu Fibonacciego, którym jest 354 224 848 179 261 915 075).

PODSUMOWANIE

- Rekurencja polega na wywołaniu przez funkcję samej siebie.
- Rozwiązanie problemu za pomocą rekurencji polega na rozkładzie na mniejsze problemy, z których można zbudować rozwiązanie problemu wyjściowego i które prowadzą w końcu do problemów z rozwiązaniem bezpośrednim.
- Zaletą rekurencji jest prosty zapis funkcji. Często wystarczy odwzorować definicję rekurencyjną w zapisie funkcji. Wady rekurencji można zauważyć dla większych danych – są związane z brakiem pamięci na przechowywanie wywołań rekurencyjnych lub duplikowaniem obliczeń. Można to zaobserwować w przypadku ciągu Fibonacciego. Rozwiązanie rekurencyjne nie sprawdza się dla większych wartości parametru n, ponieważ funkcja wielokrotnie wywołuje podproblemy i nie korzysta z raz obliczonego wyniku, tylko każde wywołanie oblicza niezależnie. Lepiej sprawdza się tutaj iteracja.
- Rekurencję można zastąpić iteracją, zwłaszcza gdy jest to rekurencja ogonowa, w której rekurencyjne wywołanie funkcji znajduje się w ostatnim wierszu.

PYTANIA SPRAWDZAJĄCE

1. Jaką funkcję nazywamy rekurencyjną?
2. Co jest zaletą rekurencji?
3. Dlaczego rozwiązanie iteracyjne jest szybsze niż rekurencja w przypadku obliczania n-tej liczby ciągu Fibonacciego?

ZADANIA DODATKOWE

Zadanie 1. Funkcja rekurencyjna

Napisz wartość oraz kolejne wywołania funkcji **`moja(x)`** dla **`x = 15`**.

```
1. def moja(x):
2.     if x > 17:
3.         return 4
4.     return moja(x + 1) + x
```

Zadanie 2. Złota proporcja

Wypisz kolejne ilorazy wartości $F(n)$ i $F(n - 1)$ ciągu Fibonacciego dla przedziału od 2 do 20. Jaką widać prawidłowość?

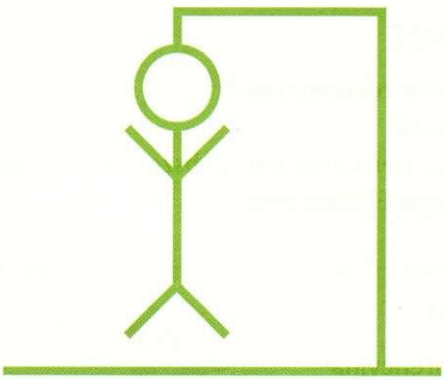
6. Przygotowanie gry

- Kolejne kroki opracowywania gry
- Pisanie i testowanie programów
- Wczytywanie danych z pliku

ZADANIE NA START

Zgadywanie słów

Wisielec (szubienica) to gra polegająca na odgadywaniu słów. Pierwszy gracz wymyśla słowo i podaje, z ilu liter się składa. Drugi gracz odgaduje poszczególne litery słowa. Wygrywa, gdy odgadnie całe słowo. Gdy poda literę, która jest zawarta w słowie, pierwszy gracz wstawia ją w odpowiednie miejsca; w przeciwnym wypadku rysuje element szubienicy. Jeżeli pierwszy gracz narysuje kompletnego „wisielca", zanim drugi odgadnie słowo, wówczas wygrywa. Zaproponuj pięć takich słów do gry w wisielca, które twoim zdaniem trudno odgadnąć. Uzasadnij swój wybór.



PROJEKTOWANIE GRY

Twoim zadaniem będzie zaimplementowanie mechanizmu gry opisanej w zadaniu na start. Będziesz pracować głównie w trybie tekstowym. Jako rozszerzenie możesz dodać wczytanie listy potencjalnych haseł z pliku oraz graficzną reprezentację „wisielca".

Ogólny schemat gry przedstawia się następująco:

1. Wylosuj słowo do odgadywania i nadaj mu nazwę **haslo**.
2. Zmiennej **odgadywane** przypisz słowo złożone z gwiazdek w liczbie równej długości słowa **haslo**.
3. Ustaw liczbę dostępnych prób na 10.

4. Dopóki **odgadywane != haslo** i liczba prób > 0 :
   - poproś użytkownika o podanie litery;
   - jeśli podana litera jest w zgadywanym słowie, wstaw w słowie **odgadywane** wszystkie jej wystąpienia;
   - jeśli podanej litery nie ma w haśle, zmniejsz liczbę dostępnych prób.
5. Jeśli odgadnięto hasło, wypisz gratulacje.
6. Jeśli przekroczono liczbę prób, poinformuj o przegranej.

WCZYTYWANIE DANYCH

W interfejsie przygotowywanym dla użytkownika powinno się wprowadzać krótkie i przejrzyste polecenia. Należy przy tym skorzystać z funkcji **input(napis)** do wczytywania danych oraz polecenia **print(napis)** do ich wypisywania. Przeanalizuj poniższe przykłady.

```
1. imie = input("Podaj swoje imię ")
2. print("Witaj,", imie, ".")
3.
4. kolor = input("Podaj ulubiony kolor ")
5. if kolor == "niebieski":
6.     print("Ja też lubię niebieski.")
7. else:
8.     print("Ty lubisz", kolor, "- ja wolę niebieski.")
```

Rys. 1. Przykłady operacji wejścia i wyjścia

Ćwiczenie 1. Opracowanie gry

Opracuj główny mechanizm gry w wisielca według podanego wcześniej schematu.

- Zdefiniuj bezparametrową funkcję **gra()** i zainicjalizuj zmienne. Do wygenerowania hasła złożonego z gwiazdek wykorzystaj funkcję **len()**, której wynikiem jest długość słowa.

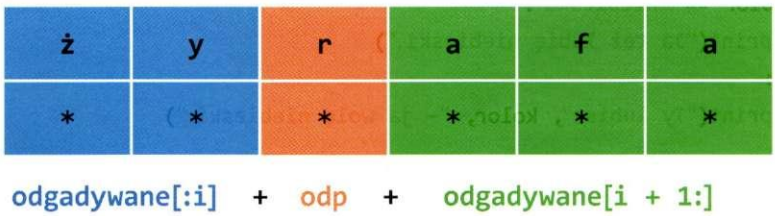
```
1. def gra():
2.     haslo = "żyrafa"
3.     odgadywane = "*" * len(haslo)
4.     proby = 10
```

- Dopisz pętlę, w której będziesz pytać użytkownika o literę, informować, ile ma prób do wykorzystania, i sprawdzać, czy litera występuje w haśle. Pętla powinna być wykonywana, dopóki hasło nie zostanie odgadnięte lub nie zostanie przekroczona liczba prób **proby > 0**.

```
1. def gra():
2.     haslo = "żyrafa"
3.     odgadywane = "*" * len(haslo)
```

```
4.      proby = 10
5.      while odgadywane != haslo and proby > 0:
6.          print("Zgadnij hasło", odgadywane)
7.          print("Pozostało prób:", proby)
8.          odp = input("Podaj literę ")
9.
10.         if odp in haslo:
11.             print("Litera", odp, "jest w haśle")
12.         else:
13.             proby -= 1
```

Jeżeli litera podana przez użytkownika występuje w haśle, trzeba ją wstawić w odpowiednie miejsca. Ponieważ w języku Python nie ma możliwości modyfikowania napisu, trzeba zbudować nowy napis z jego fragmentów. Poniższy przykład pokazuje sytuację, w której użytkownik podał literę *r*, występującą w haśle. Nowe słowo składa się ze znaków występujących do litery *r* (kolor niebieski), litery *r* (kolor pomarańczowy) oraz liter występujących po literze *r* (kolor zielony). Jeśli indeks litery *r* oznaczy się przez **i**, to nowe słowo można zapisać w następującej postaci:



Rys. 2. Budowa nowego napisu z jego fragmentów

Notacja **napis[i : j]** oznacza wyodrębnienie fragmentu napisu od **i** do **j - 1** włącznie. Jeżeli pierwszy argument jest pomijany, to domyślnie jest to 0, jeśli ostatni – to domyślnie jest to długość napisu.

Ćwiczenie 2. Podmiana litery

Uzupełnij skrypt z ćwiczenia 1 o podmianę litery.

- Zmodyfikuj wiersz 11. W pętli **for** należy przeglądać hasło litera po literze i sprawdzać, czy litera podana przez użytkownika w nim występuje. Jeśli tak, trzeba podmienić gwiazdkę znajdującą się pod danym indeksem na literę.
  - Zwróć uwagę na dwukropki, które wyodrębniają fragment napisu – ich pominięcie będzie prowadzić do błędu.

```
11.         for i in range(len(haslo)):
12.             if odp == haslo[i]:
13.             odgadywane = odgadywane[:i] + odp + odgadywane[i + 1:]
```

- Dopisz zakończenie.

```
    if haslo == odgadywane:
        print("Gratulacje - hasło to", haslo)
    else:
        print("Nie odgadnięto hasła :-(")
```

- Przetestuj całość. Zmodyfikuj hasło do odgadnięcia.

```
def gra():
    haslo = "żyrafa"
    odgadywane = "*" * len(haslo)
    proby = 10
    while odgadywane != haslo and proby > 0:
        print("Zgadnij hasło", odgadywane)
        print("Pozostało prób: ", proby)
        odp = input("Podaj literę ")

        if odp in haslo:
            for i in range(len(haslo)):
                if odp == haslo[i]:
                odgadywane = odgadywane[:i] + odp + odgadywane[i + 1:]
        else:
            proby -= 1

    if haslo == odgadywane:
        print("Gratulacje - hasło to", haslo)
    else:
        print("Nie odgadnięto hasła :-(")
```

WCZYTYWANIE DANYCH Z PLIKU

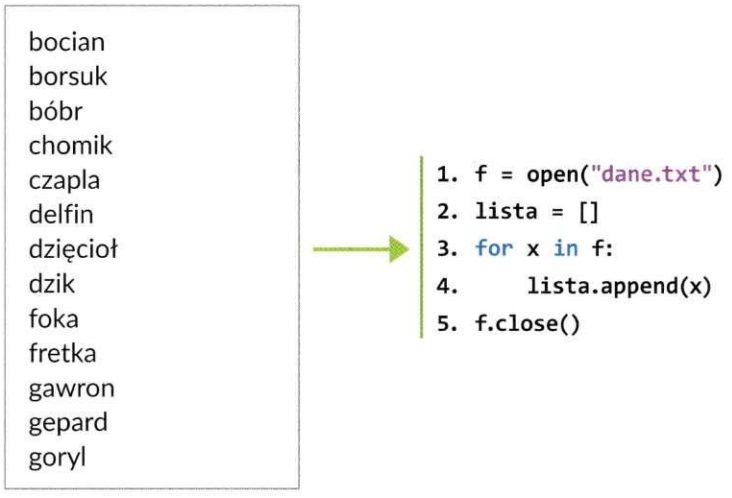
Aby dodać możliwość losowania hasła, można po prostu podać listę słów w programie. Należy wówczas zdefiniować listę **slownik** i umieścić ją na początku funkcji, a następnie dopisać losowanie. W tym celu trzeba zaimportować z modułu **random** funkcję **choice**, której wynikiem jest losowo wybrany element listy.

```
from random import choice
def gra():
    slownik = ["żyrafa", "słoń", "nosorożec", "szympans", "aligator"]
    haslo = choice(slownik)
    odgadywane = "*" * len(haslo)
    proby = 10
```

Rys. 3. Początek funkcji po wprowadzonych zmianach

W przypadku większej ilości danych warto zapisać je w pliku, a w trakcie działania programu wczytać do pamięci. W ten sposób funkcjonalność programu jest oddzielana od danych potrzebnych do jego przetwarzania.

Należy w tym celu utworzyć plik tekstowy z hasłami zapisanymi jedno pod drugim, a następnie wprowadzić zawarte w nim dane do programu – tj. otworzyć plik, odczytać jego zawartość (poniżej wiersz 1) i zapamiętać w zmiennej (wiersze 3 i 4), po czym zamknąć plik (wiersz 5).



początek pliku tekstowego **dane.txt** kod wczytujący plik tekstowy **dane.txt**

Rys. 4. Wczytywanie danych z pliku tekstowego

Na liście zostaną zapisane dane wiersz po wierszu, łącznie z białymi znakami (np. spacjami) i końcami wiersza, dlatego lepiej jest pozbyć się tych znaków za pomocą metody **strip()**.

```
1. f = open("dane.txt")
2. lista = []
3. for x in f:
4.     lista.append(x.strip())
5. f.close()
```

Rys. 5. Wczytywanie haseł z pliku tekstowego i jednoczesne usunięcie niepotrzebnych znaków

Ćwiczenie 3. Wiele haseł

Dodaj do skryptu z ćwiczenia 2 losowanie hasła przez wczytanie pliku z hasłami.

- Zdefiniuj funkcję **wczytaj(nazwa)**, której wynikiem jest zawartość pliku o nazwie podanej jako argument.

```
1. def wczytaj(nazwa):
2.     f = open(nazwa)
3.     lista = []
4.     for x in f:
5.         lista.append(x.strip())
6.     f.close()
7.     return lista
```

- Przygotuj plik tekstowy **zwierzeta.txt** z nazwami zwierząt do odgadywania. Zapisz go w tym samym katalogu.
- Uzupełnij funkcję **gra()** o wczytywanie losowego hasła z pliku. Zaimportuj z modułu **random** funkcję **choice**, której wynikiem jest losowo wybrany element listy.

```
1. from random import choice
2. def gra():
3.     slownik = wczytaj("zwierzeta.txt")
4.     haslo = choice(slownik)
5.     odgadywane = "*" * len(haslo)
6.     proby = 10
```

GRAFIKA

Jak już wiesz, do rysowania w Pythonie można wykorzystać moduł **turtle**. Zapewne pamiętasz polecenia: **fd(kroki)**/**bk(kroki)** – idź naprzód lub wstecz o podaną liczbę kroków, **rt(kąt)**/**lt(90)** – obróć się w lewo lub w prawo o podany kąt, **pu()**, **pd()** – podnieś pisak lub opuść pisak. Przetestuj działanie kolejnych poleceń zaprezentowanych w poniższej tabeli.

Polecenie	Znaczenie	Opis działania
st()	*show turtle*	Pokaż żółwia.
ht()	*hide turtle*	Schowaj żółwia.
clear()	*clear*	Wyczyść ekran
seth(kąt)	*set heading*	Skieruj żółwia w określonym kierunku.
sethpos(pozycja)	*set position*	Przemieść żółwia na podaną pozycję.

Ćwiczenie 4. Rysowanie szubienicy

Dodaj do skryptu z ćwiczenia 3 rysowanie szubienicy z wisielcem w zależności od stanu gry.

- Przygotuj funkcję **rysunek()**, po wywołaniu której powstanie rysunek szubienicy z wisielcem. W razie potrzeby przeanalizuj komentarze w podanym kodzie.

```
1. from turtle import *
2. def rysunek():
3.     #czyszczenie
4.     seth(0); setpos(0, 0); clear()
5.     #podstawa
6.     bk(80); fd(40); lt(90)
7.     #pion
8.     fd(200); rt(90)
9.     #poziom
10.    fd(100); rt(90)
11.    #haczyk
12.    fd(30)
13.    #głowa
14.    lt(90)
15.    for i in range(36):
16.        fd(3); rt(10)
17.    rt(90)
18.    #tułów
19.    pu(); fd(36); pd()
20.    fd(60)
21.    #prawa ręka
22.    bk(50); rt(45); fd(30); bk(30)
23.    #lewa ręka
24.    lt(90); fd(30); bk(30); rt(45)
25.    #prawa noga
26.    fd(50); rt(30); fd(50); bk(50)
27.    #lewa noga
28.    lt(60); fd(50); bk(50); rt(45)
29.    ht()
```

- Uzupełnij rysunek o możliwość rysowania częściowej szubienicy w zależności od stanu gry. Przeanalizuj poniższy kod i dodane do niego komentarze.

```
1. def rysunek(nr):
2.     #czyszczenie
3.     seth(0); setpos(0, 0); clear()
4.     if nr == 10: return
5.
6.     #podstawa
7.     bk(80); fd(40); lt(90)
8.     if nr == 9: return
```

```
9.
10.     #pion
11.     fd(200); rt(90)
12.     if nr == 8: return
13.
14.     #poziom
15.     fd(100); rt(90)
16.     if nr == 7: return
17.
18.     #haczyk
19.     fd(30)
20.     if nr == 6: return
21.
22.     #głowa
23.     lt(90)
24.     for i in range(36):
25.         fd(3); rt(10)
26.     rt(90)
27.     if nr == 5: return
28.
29.     #tułów
30.     pu(); fd(36); pd()
31.     fd(60)
32.     if nr == 4: return
33.
34.     #prawa ręka
35.     bk(50); rt(45); fd(30); bk(30)
36.     if nr == 3: return
37.
38.     #lewa ręka
39.     lt(90); fd(30); bk(30); rt(45)
40.     if nr == 2: return
41.
42.     #prawa noga
43.     fd(50); rt(30); fd(50); bk(50)
44.     if nr == 1: return
45.
46.     #lewa noga
47.     lt(60); fd(50); bk(50); rt(45)
48.     ht()
```

ASCII ART ZAMIAST GRAFIKI

Rysowanie z biblioteką **turtle** można zastąpić rysunkami ze znakami ASCII.

```
                                                +===+
                          |                     |   |
                          |                     |   o
        ----->            |        ----->       |  /|\
                          |                     |  / \
                          |                     |
=========           =========           =========
```

Rys. 6. Przykładowe rysunki szubienicy i wisielca, wykonane za pomocą znaków ASCII

W grze funkcja „rysująca" szubienicę i wisielca może korzystać z polecenia **print()** i wypisywania znaków. Trzeba tylko zdefiniować kolejne rysunki i pamiętać o tym, że w Pythonie napisy wielowierszowe oznacza się na początku i końcu przez potrójny cudzysłów **'''**, a znak **** trzeba wypisać dwukrotnie, gdyż jest to znak dosłowności.

```
def wisielec(proba):
    if (proba == 10):
        return '''''========='''
    if (proba == 9):
        return '''''
|
|
|
|
|
========= '''
    return '''''
+===+
|   |
|   o
|  /|\\
|  / \\
|
========='''
```

Rys. 7. Fragment funkcji wisielec (funkcję należy uzupełnić o brakujące „rysunki")

Ćwiczenie 5. Wisielec ze znakami ASCII

Zmodyfikuj skrypt z ćwiczenia 3 tak, aby wyrysować szubienicę i wisielca za pomocą znaków ASCII.

PODSUMOWANIE

- Przy projektowaniu gry warto przemyśleć i uściślić zasady, rozpisać schemat gry, a potem krok po kroku go implementować. Na koniec trzeba przetestować grę. Z czasem można rozwijać różne funkcjonalności, by gra była ciekawsza.
- Aby implementować grę, należy mieć podstawowe umiejętności programistyczne. Wskazane jest wykorzystanie zmiennych i bardziej złożonych struktur danych, np. list. Przydatne są też funkcje, pętle oraz instrukcje warunkowe.
- Jeśli potrzeba więcej danych, warto zapisać je w pliku, a w trakcie działania programu wczytać do pamięci. Przed rozpoczęciem czytania należy plik otworzyć (**open**), a na koniec zamknąć (**close**).
- Do przygotowania prostej grafiki można użyć znaków. Pomocna może okazać się znajomość kodów ASCII, lepiej jednak korzystać z możliwości graficznych programów.

PYTANIA SPRAWDZAJĄCE

1. Od czego należy zacząć tworzenie gry?
2. W jaki sposób można modyfikować napisy w Pythonie?
3. Jakich poleceń użyjesz do czytania danych z pliku?

ZADANIA DODATKOWE

Zadanie 1. Język angielski

Rozszerz grę o odgadywanie słów w języku angielskim. Dodaj możliwość wyboru pliku, z którego są wczytywane hasła, oraz języka komunikatów.

Zadanie 2. Nowe funkcjonalności

Zaproponuj i zaimplementuj nowe funkcjonalności, np. zliczanie wygranych i przegranych, możliwość wprowadzania całego hasła lub tworzenia haseł wielowyrazowych. Wprowadzone zmiany przetestuj na forum klasy.

Zadanie 3. Wisielec i Pygame

Poszukaj implementacji gry Hangman dla biblioteki Pygame i przetestuj ją na swoim komputerze.

Wskazówka

Bibliotekę Pygame pobiera się w postaci pakietu za pomocą instalatora pakietów **pip**.

- Instalator powinien być dołączony podczas instalacji środowiska Python, w przeciwnym razie trzeba doinstalować go ręcznie (sposób postępowania zależy od

systemu operacyjnego i jest opisany na stronie **https://www.pygame.org/wiki/GettingStarted**).

- Aby pobrać bibliotekę Pygame, należy uruchomić tekstowe okno konsoli i wpisać polecenie zawierające nazwę pakietu – instalator sam wyszuka w internecie potrzebny plik: **Start → Uruchom → `cmd` → `pip install pygame`**.