Laboratorium 9

Schematy blokowe i algorytmy

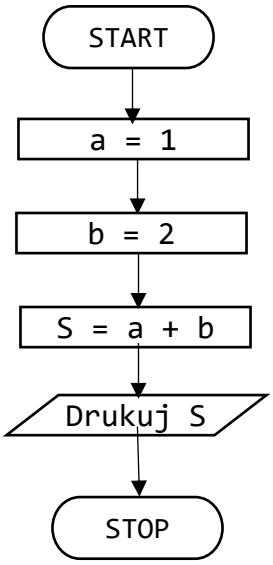
Schematy blokowe

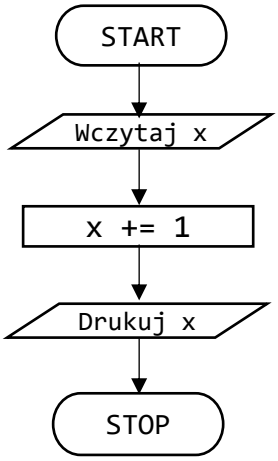
Do modelowania algorytmów (programów) często stosuje się schematy blokowe. Podstawowe bloki to:

Blok przetwarzania (procesu) | Blok danych: (pobieranie lub drukowanie) | Blok START/STOP | Blok decyzyjny (warunek)

Poniżej przykładowe zastosowania:

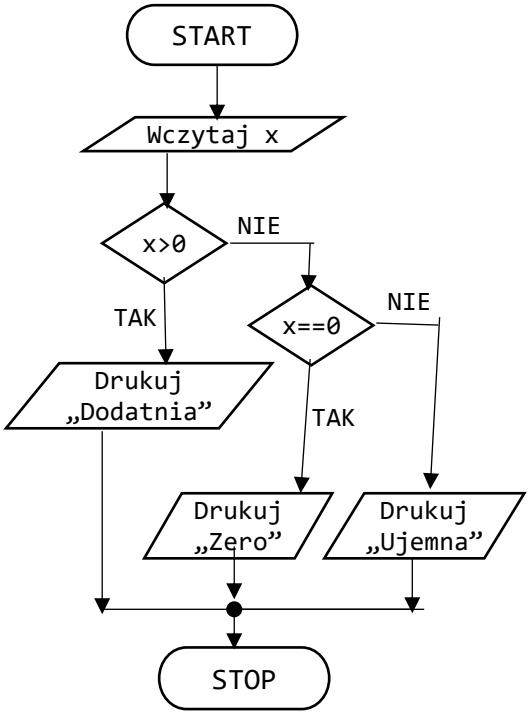
Program	Schemat	Co wypisze?

`a = 1`
`b = 2`
`S = a + b`
`print(S)`
		3

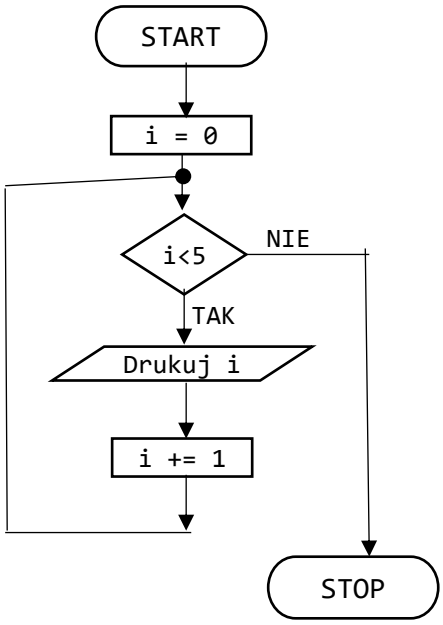
`x = int(input())`
`x += 1`
`print(x)`
		Program wczyta liczbę x i wypisze liczbę o jeden większą.

<table>
<tr>
<td>

```
x = int(input())
if x > 0:
    print("Dodatnia")
elif x == 0:
    print("Zero")
else:
    print("Ujemna")
```

</td>
<td> 
</td>
<td>Program wczyta liczbę x i wypisze „Ujemna”, „Dodatnia”, lub „Zero” w zależności od znaku x</td>
</tr>
<tr>
<td>

```
i = 0
while i < 5:
    print(i)
    i += 1
```

</td>
<td> 
</td>
<td>0
1
2
3
4</td>
</tr>
<tr>
<td>

```
for i in range(5):
    print(i)
```

</td>
<td>Schemat jak wyżej (pętla for jest realizowana jak while)</td>
<td>Wynik jak wyżej</td>
</tr>
</table>

Zadanie 1

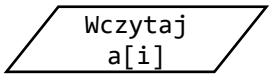
Napisz program, który wypisze liczby naturalne od 0 do 10. Gdy liczba jest parzysta to dodatkowo wypisz „Parzysta”, a gdy nieparzysta to dodatkowo wypisze „Nieparzysta”.

Skonstruuj schemat blokowy dla tego programu. Możesz użyć Worda i opcji wstawiania kształtów, programu graficznego, albo dedykowanych narzedzi do tworzenia schematów blokowych (Flowchart) np. https://www.zenflowchart.com/

Zadanie 2

Napisz program, który spyta się użytkownika o 10 liczb, zapisze je na liście `a`. Następnie wydrukuje tę listę od elementu ostatniego, a na końcu wydrukuje element maksymalny z listy.

Uwaga, wczytywanie listy zrealizujemy za pomocą pętli for, która po kolei będzie wyczytywała elementy a[0], a[1],....

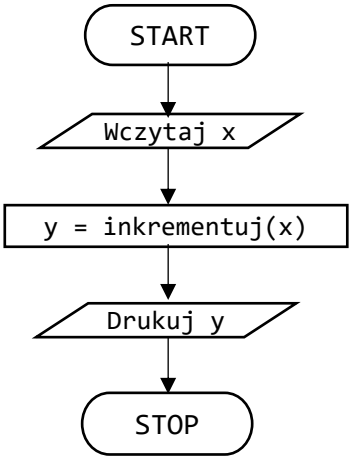


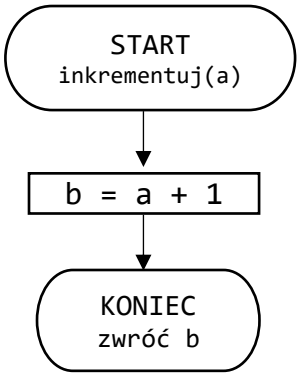
Drukowanie listy też należy zrealizować w pętli. Za każdym razem drukujemy tylko jeden element.

Skonstruuj schemat blokowy dla tego programu uwzględniający obie pętle.

Zadanie 3

Obok schematu dla programu głównego, można zrobić schemat blokowy dla funkcji. Nazwę funkcji z argumentem umieszczamy w bloku startowym, a zwracaną wartość (o ile jest jakaś zwracana), umieszczamy w bloku końcowym (dla funkcji piszemy KONIEC zamiast STOP).



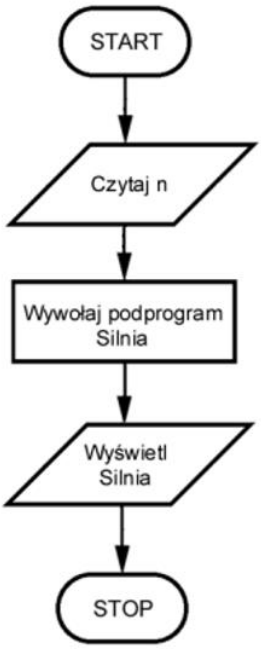


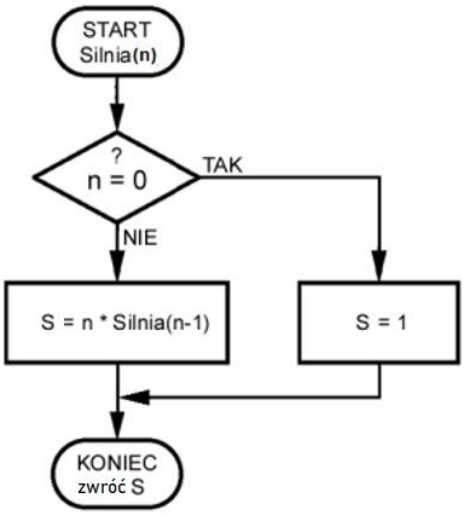
a) Skonstruuj schemat blokowy dla programu głównego i dla funkcji obliczającej pole prostokąta (z badaniem czy boki są dodatnie): `rectangle_area(a, b).`

Zadanie 4

Ostatnio realizowaliśmy silnię metodą rekurencyjną. Przypomnij sobie te program. Załóżmy, że wczytujemy n i chcemy obliczyć n!. Piszemy funkcję silnia(n), która ma zwrócić n!.

Musimy stworzyć schemat blokowy dla głównego programu i dla funkcji silnia.





Zauważ, że z uwagi na rekurencję. W środku funkcji Silnia, korzystamy z funkcji Silnia.

b) Skonstruuj schemat blokowy dla rekurencyjnego programu obliczającego n-ty element ciągu Fibonacciego. Program rozbij na schemat blokowy dla programu głównego i schemat blokowy dla funkcji.

Zadanie 5

Sortowanie bąbelkowe służy do sortowania (rosnącego lub malejącego) tablic z liczbami.

Załóżmy, że chcemy posortować tablicę rosnąco (od najmniejszej do największej). Zasada jest taka: przechodzimy przez tablicę od lewej do prawej i wypychamy największą liczbę na ostatnie miejsce w tablicy dokonując porównań par elementów. Przypomina to zachowanie bąbelków w wodzie: największe bąbelki najszybciej lecą do góry. Następnie znowu idziemy od lewej do prawej i wypychamy drugą największa liczbę na przedostatnie miejsce. Procedurę powtarzamy, aż zabraknie elementów do wypychania.

Pseudokod:

```
bubblesort(a):
  for i=0 to length(a)-1:
    for j=0 to length(a)-i-1:
      if a[j] > a[j+1]:
        zamień(a[j], a[j+1])
```

a) Zasymuluj krok po kroku ten algorytm dla listy `a = [6, 2, 7, 1, 5]`. Wykorzystaj do tego załączony plik `bubblesort.xlsx`. Uzupełnij tabelkę

(kolorowe pola) elementami tablicy a w trakcie sortowania tzn. po pojedynczych przebiegach pętli `for` z `i` i pętli `for` z `j`. Zapisz plik i pokaż go podczas sprawdzania.

i	j	a[0]	a[1]	a[2]	a[3]	a[4]
		6	2	7	1	5
0	0					
0	1					
0	2					
0	3					
1	0					
1	1					
1	2					
2	0					
2	1					
3	0					

b) Napisz tę funkcję w pythonie i przetestuj ją dla powyższej 5-elementowej tablicy. Wypisz posortowaną tablicę.
c) Zrób wersję funkcje `bubblesort_min(a)`, która sortuje malejąco. Przetestuj ją. Wypisz posortowaną tablicę.
d) Skonstruuj schemat blokowy dla tego programu.

Zadanie 6

Jak obliczamy najmniejszy wspólny dzielnik dwóch liczb (NWD) za pomocą algorytmu Euklidesa? Poczytaj o tym w Internecie np. tutaj:
http://www.algorytm.edu.pl/algorytmy-maturalne/algorytm-eulkidesa.html

Napisz algorytm w wersji zoptymalizowanej metodą rekurencyjną i nierekurencyjną. Dla obu wersji podaj schematy blokowe.

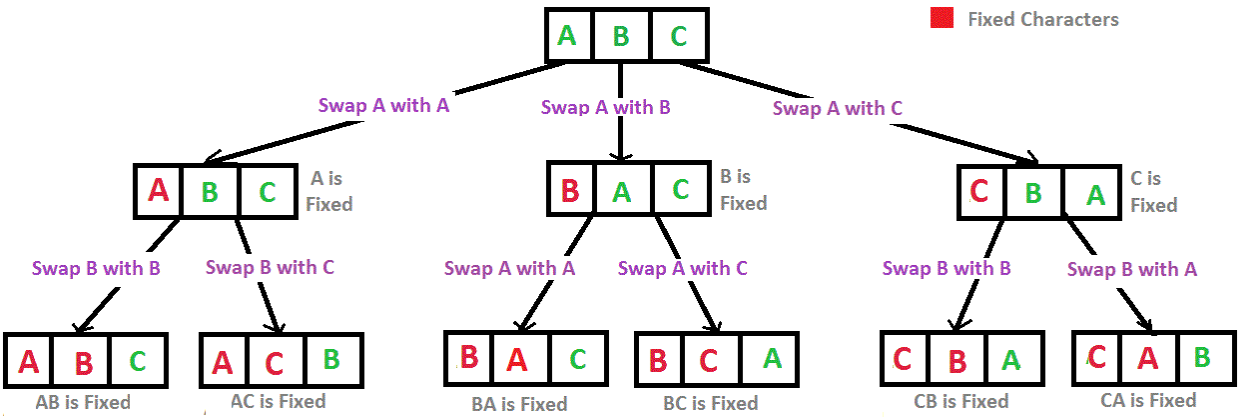
Zadanie 7

Poniżej przedstawiono algorytm do generowania wszystkich permutacji elementów listy.

```
def permute(a, l, r):
    if l == r:
        print(a)
    else:
        for i in range(l, r+1):
            a[l], a[i] = a[i], a[l]
            permute(a, l+1, r)
            a[l], a[i] = a[i], a[l]


a = ['A', 'B', 'C']
permute(a, 0, len(a)-1)
```

Program korzysta z rekurencyjnej funkcji `permute`. Rekurencja polega na ustaleniu pierwszego elementu permutacji, i uruchomieniu `permute` dla reszty elementów. Dobrze widać to na poniższym drzewie rekursji:



Recursion Tree for Permutations of String "ABC"

a) Uruchom program i sprawdź, jak działa.

b) Zmień program tak, aby zamiast wypisywania permutacji, była ona dodawana do listy wyników, zmienna globalna `results = []`
Po wywołaniu funkcji `permute`, wyświetl zawartość zbioru `results`.

Zadanie 8

Mamy tutaj inną wersję programu do generowania permutacji. Wykorzystującą `return`.

```
def permutation(lst):
    if len(lst) == 0:
        return []
    if len(lst) == 1:
        return [lst]
    perms = []
    for i in range(len(lst)):
       m = lst[i]
       remLst = lst[:i] + lst[i+1:]
       for p in permutation(remLst):
           perms.append([m] + p)
    return perms


print(permutation(['A', 'B', 'C']))
```

a) Uruchom program i sprawdź wynik działania.
b) Skorzystaj z [Recursion Tree Visualizer] i zobacz jakie zostanie wygenerowane drzewo. Zapisz je jako obrazek.

   Następnie w komentarzach pod programem zapisz własnymi słowami odpowiedzi na pytania:

c) Co oznaczają poniższe linijki kodu? Do czego służą?

```
    if len(lst) == 0:
        return []
    if len(lst) == 1:
        return [lst]
```

d) Co jest przechowywane w obiekcie `lst`? A co w obiekcie `perms`?

e) Co się dzieje w poniższych linijkach kodu? Co to są `m` i `remLst`?

```
    m = lst[i]
    remLst = lst[:i] + lst[i+1:]
```

f) Podaj przykład `lst`, `i`, `m` i `remLst` po przejściu przez te dwie linijki kodu.

g) Do czego służy poniższa pętla `for`? Jak rozumiesz sklejenie `[m]+p`?

```
for p in permutation(remLst):
            perms.append([m] + p)
```

Zadanie 9

Poniżej znajduje się program na wygenerowanie wszystkich możliwych podzbiorów zbioru (listy). Program ma jedną lukę, którą trzeba uzupełnić: (...).

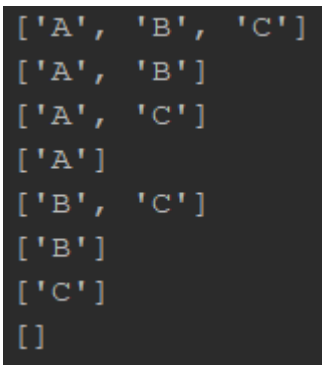
```
def subsets(a, i, sub):
    if i == len(a):
        print(sub)
    else:
        subsets(a, i+1, sub + [a[i]])
        subsets(...)


a = ['A', 'B', 'C']
subsets(a, 0, [])
```

Idea działania programu:

- `a` przetrzymuje całą listę
- indeks `i` wskazuje na aktualny element, zaczyna się od 0 i rośnie wraz z wywoływaniem funkcji rekurencyjnej
- parametr `sub` przetrzymuje aktualnie tworzony podzbiór
- rekurencja działa na zasadzie rozpatrzenia dwóch przypadków: albo wrzucamy `i`-ty element do zbioru `sub`, albo go nie wrzucamy. Rozpatrzenie wszystkich takich możliwości stworzy wszystkie możliwe podzbiory.

a) Uzupełnij brakujący fragment kodu pythonowego (tak gdzie znajduje się wielokropek)
Sprawdź czy program generuje wszystkie podzbiory:



b) Narysuj drzewo rekursji dla programu i a = ['A', 'B', 'C'] (podobnie jak w zadaniu 7). W węzłach drzewa, powinien znajdować się element sub. Dołącz obrazek drzewa do rozwiązania.