

Paradygmaty programowania

# WPROWADZENIE

***dr inż. Łukasz Bartczuk***

Katedra Inteligentnych Systemów Informatycznych

p. 517

lukasz.bartczuk@pcz.pl

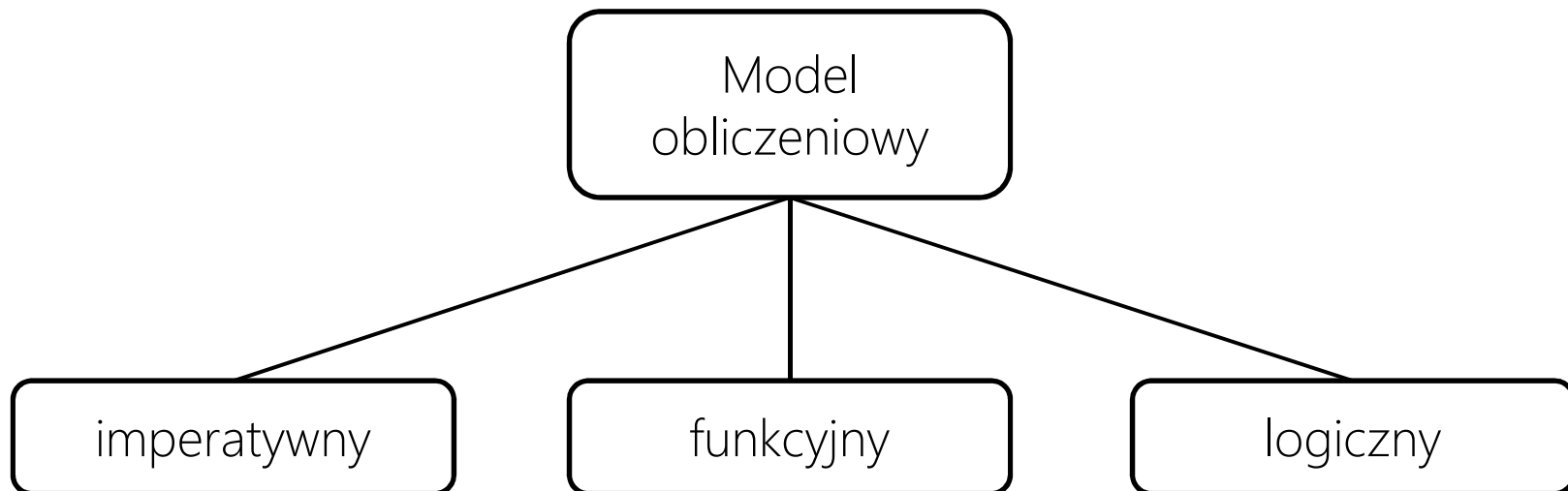
Co to jest komputer?

Co to jest  
program komputerowy?

# Model obliczeniowy

---

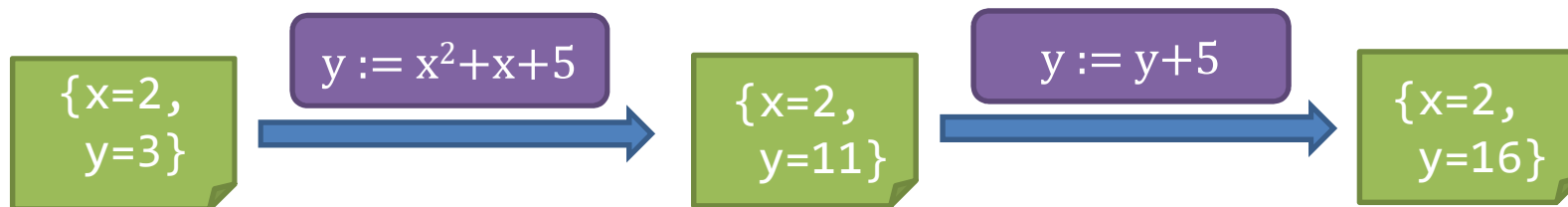
Model obliczeniowy jest to zbiór wartości (prostych lub złożonych), skojarzonych z nimi operacji oraz operacji wykorzystywanych do zdefiniowania obliczeń.



# Model imperatywny

---

Model imperatywny składa się wartości oraz stanu i operacji przypisania.

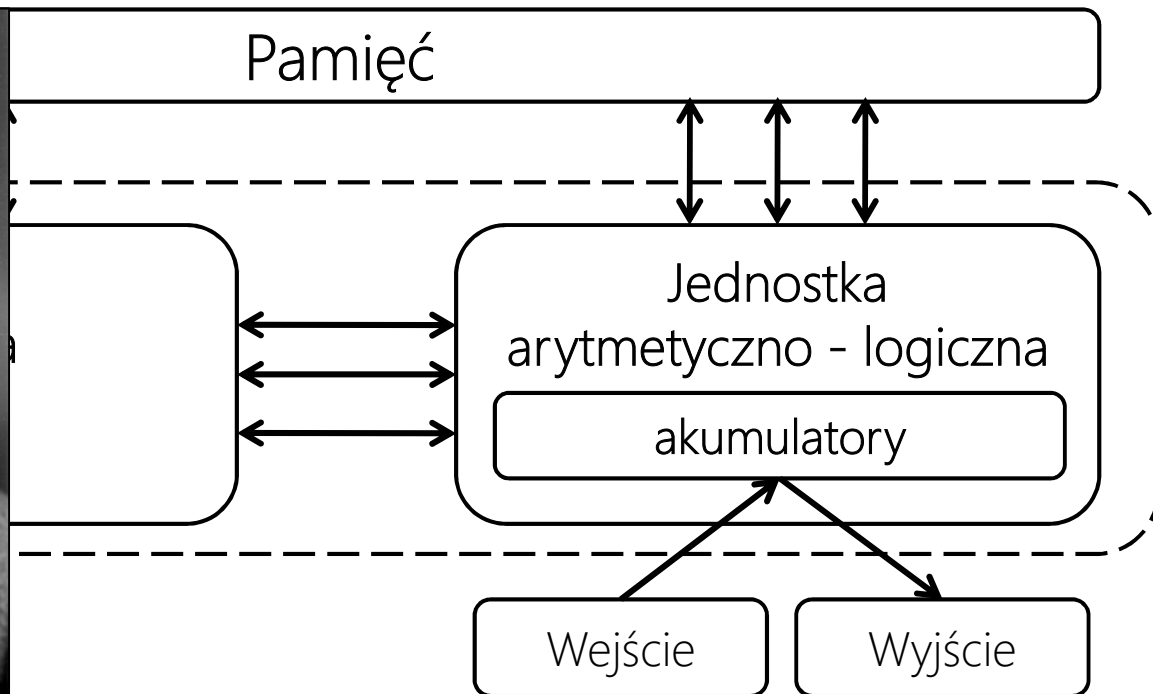


# Maszyna John'a von Neumanna

---



1903-1957





# Model funkcyjny

---

Model ten składa się z wartości i funkcji (przy czym funkcje też są wartością), a podstawową operacją do wykonywania obliczeń jest aplikacja funkcji do wartości.

$$f(x) = x^2 + x + 5$$

$$g(y) = y + 5$$



# Model logiczny

---

Model ten składa się z faktów, relacji i zapytań, a metodą dokonywania obliczeń jest wnioskowanie logiczne.

1. `student(tomek).`
2. `LubiParadygmaty(X)` jeżeli `student(X)`.
3. ?- `LubiParadygmaty(Y)`.

---

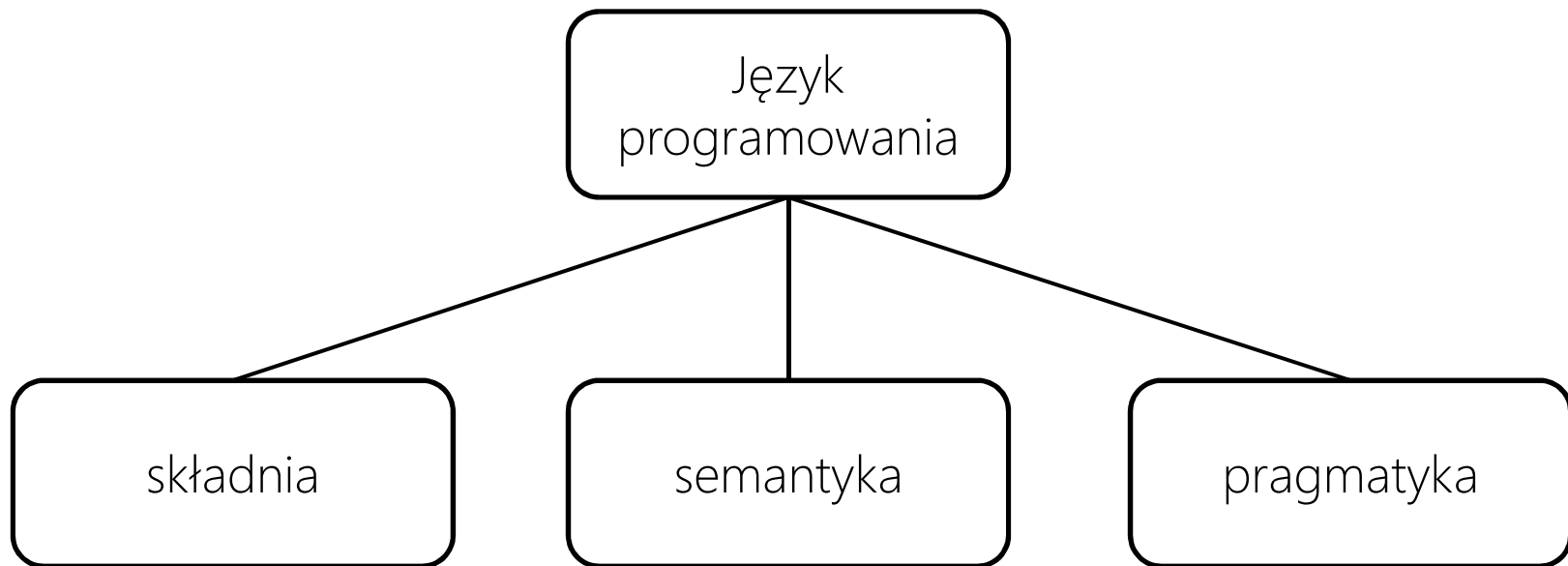
4. `student(Y)`.

---

5.  $Y = \text{tomek}$ .

# Co to jest język programowania?

---



# Składnia języków programowania

---

Zbiór zasad określających, który ciąg znaków jest poprawną instrukcją w danym języku programowania.

Do definiowania składni języka programowania najczęściej używa się notacji BNF (Backus-Naur Form)

```
<expression> ::= <term> |  
                <expression> '+' <term> |  
                <expression> '-' <term>  
<term>         ::= <factor> |  
                <term> '*' <factor> |  
                <term> '/' <factor>  
<factor>       ::= <number> | '(' <expression> ')'
```

# Semantyka języków programowania

---

Określa relację pomiędzy elementami składni języka programowania, a modelem obliczeniowym.

Semantyka aksjomatyczna

Semantyka denotacyjna

Semantyka operacyjna

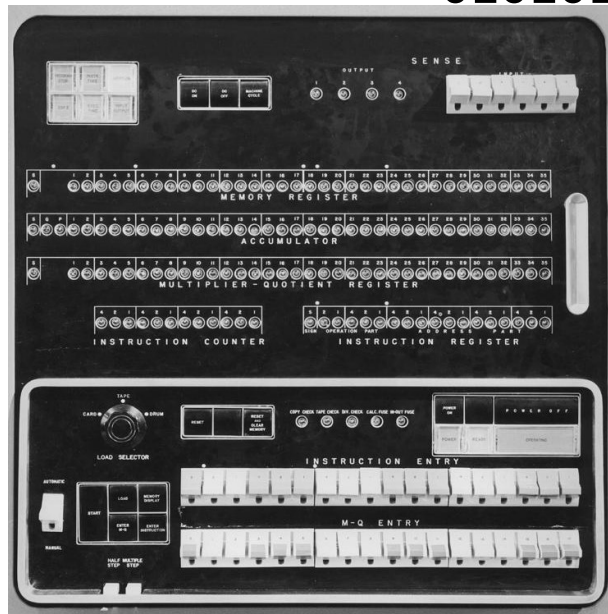
# Pragmatyka języków programowania

---

Opisuje praktyczne aspekty implementacji języka i korzystania z niego. Określa m.in. jak konstrukcje i cechy języka mogą być wykorzystane do osiągnięcia określonego celu.

# Ewolucja języków programowania

Początkowo programy wyglądały tak:



010101011000100

000001000

000000010

001100011

100000000

100100010

01111101001000110100000100

10111101000100010111111100

00000000000000000000000000000000

00110010011100001110010000

The image shows an IBM Calculator Instruction Card. It is a long, narrow card with multiple columns. The columns are labeled: IDENTIFICATION, LOCATION, ADDRESS PART, SYMBOLIC LOCATION, OPERATION PART, SYMBOLIC ADDRESS PART, and COMMENT. The card contains a series of numbers and symbols, likely representing a program or data. The card is yellowed and shows signs of age.

# Ewolucja języków programowania

---

Funkcja w języku assembler  
(nowoczesnym)



1919 - 2001

```
push ebp
mov ebp, esp
sub esp, 0x10
mov DWORD PTR [ebp-0xc], 0x5
mov DWORD PTR [ebp-0x8], 0x6
mov eax, DWORD PTR [ebp-0x8]
mov edx, DWORD PTR [ebp-0xc]
lea eax, [edx+eax*1]
mov DWORD PTR [ebp-0x4], eax
mov eax, 0x0
leave
ret
```



# Ewolucja języków programowania

---

... i w C

```
int main()
{
    int a = 5;
    int b = 6;
    int c = a+b;
    return 0;
}
```

# Ewolucja języków programowania

---

Twórca pierwszego  
kompilatora

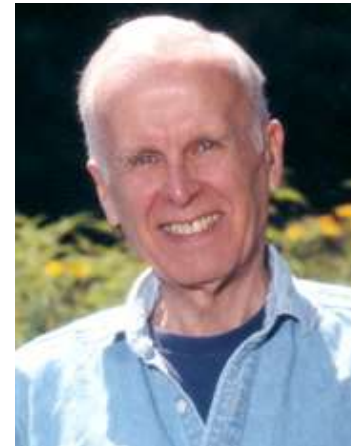
kadm. Grace Hooper



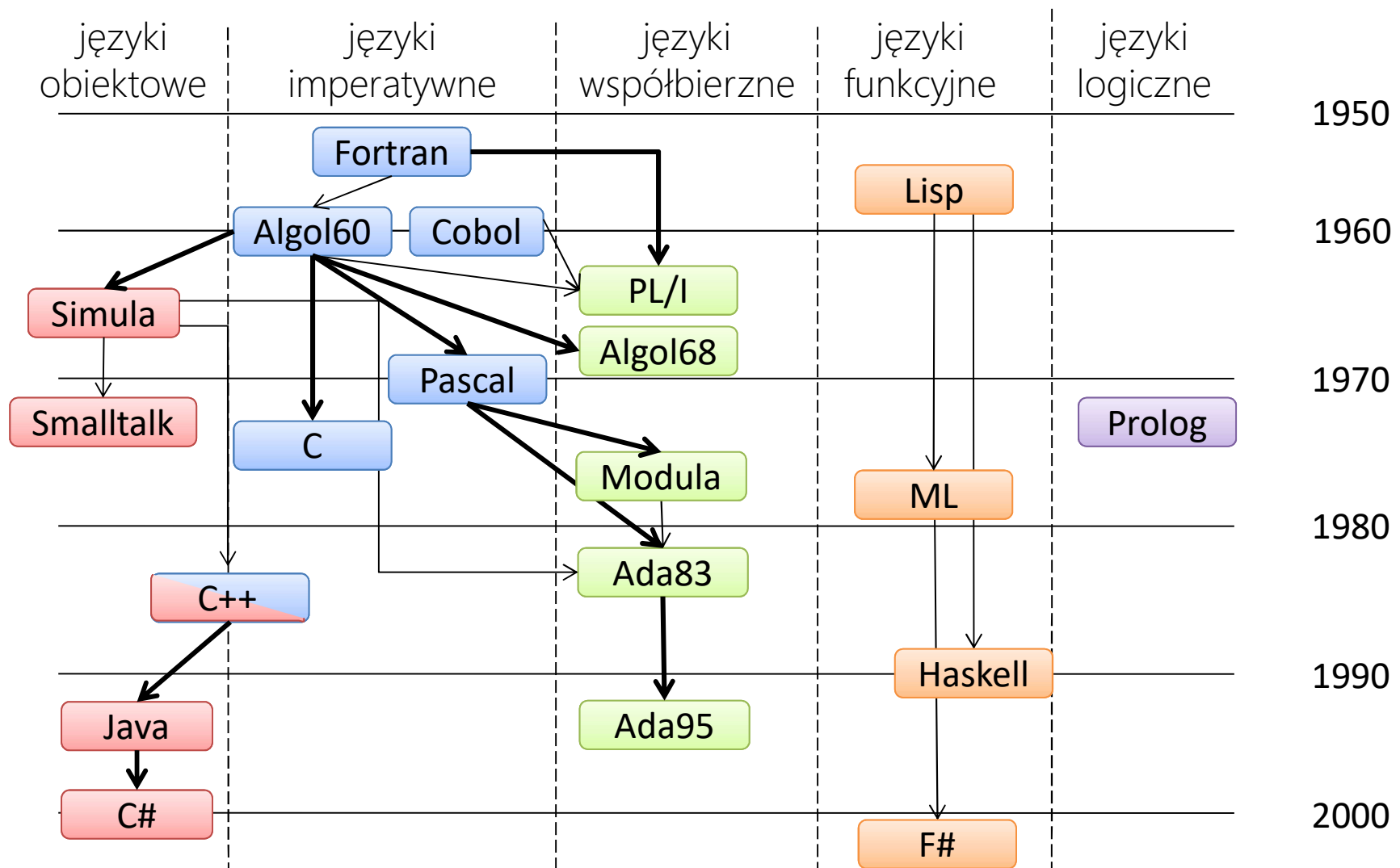
1906 - 1992

Twórca języka  
Fortran

John Backus



1924 - 2007



# Paradygmat programowania

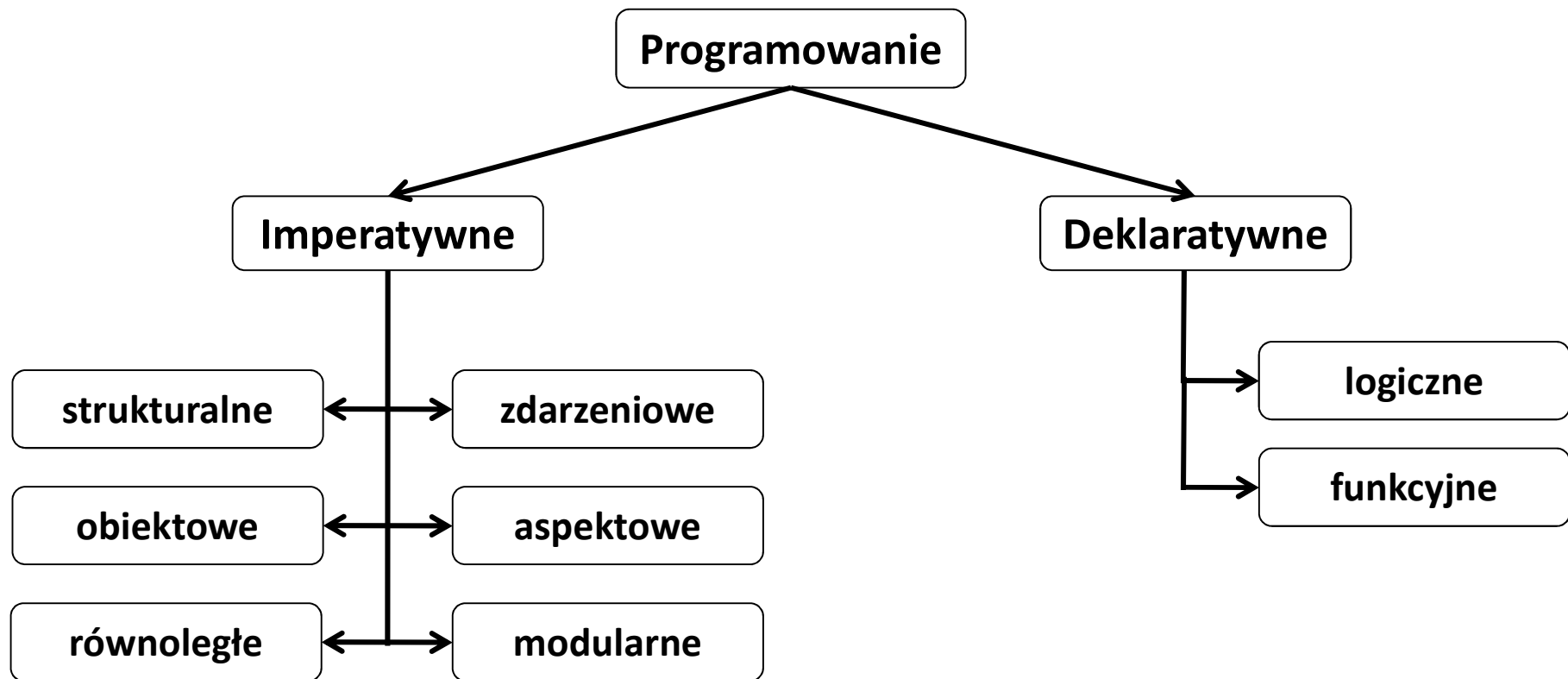
---

Paradygmat programowania jest wyróżniającym się stylem programowania. Każdy paradygmat jest scharakteryzowany przez dominację pewnych kluczowych koncepcji.

- zbiór koncepcji reprezentujących podejście do implementacji algorytmów
- zbiór mechanizmów używanych przez programistę do pisania programów i określających jak te programy będą następnie wykonywane przez komputer

# Paradygmat programowania

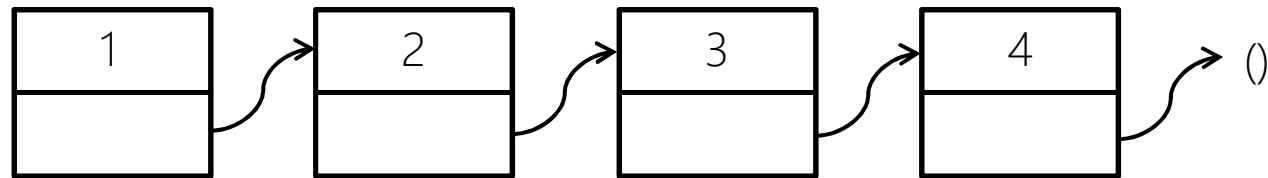
---



# Problem do rozwiązania

---

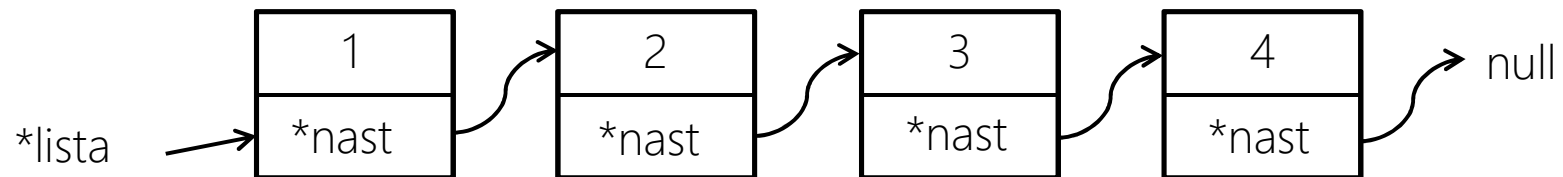
Dana jest lista liczb całkowitych:



Napisać funkcję, która będzie obliczała sumę elementów na liście

# Programowanie imperatywne

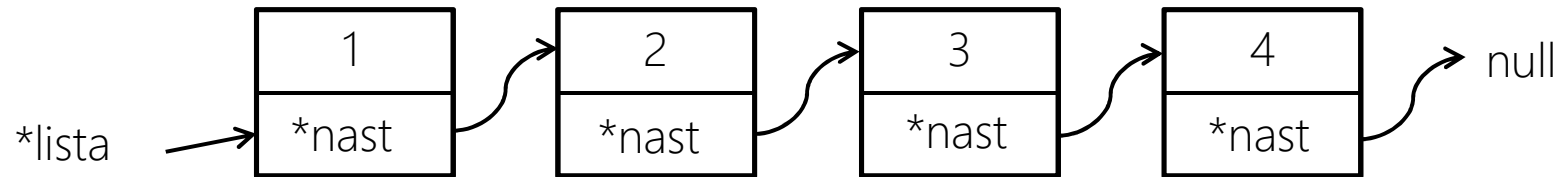
---



1. Stwórz zmienną **biezacy** określającą aktualny element listy i przypisz do niej adres pierwszego elementu na liście (parametr funkcji)
2. Stwórz zmienną **suma** przechowującą sumę wartości kolejnych elementów listy
3. W pętli (dopóki zmienna **biezacy** ma inną wartość niż **null**):
  1. Pobierz wartość składowej **wartosc** aktualnego elementu listy, pobierz wartość zmiennej **suma** zawierającą sumę wszystkich dotychczasowych elementów. Dodaj te dwie wartości do siebie. Wynik przypisz do zmiennej **suma**.
  2. Pobierz wartość wskaźnika **nast** aktualnego elementu na liście i przypisz go do zmiennej **biezacy**
4. Zwróć wartość zmiennej **suma** do programu wywołującego

# Programowanie imperatywne

---

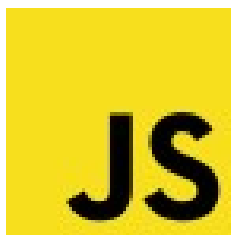


```
int Sumuj(Lista* lista) {  
    Lista* biezacy = lista;  
    int suma = 0;  
    while(biezacy != NULL) {  
        suma = suma + biezacy->wartosc;  
        biezacy = biezacy->nast;  
    }  
    return suma;  
}
```



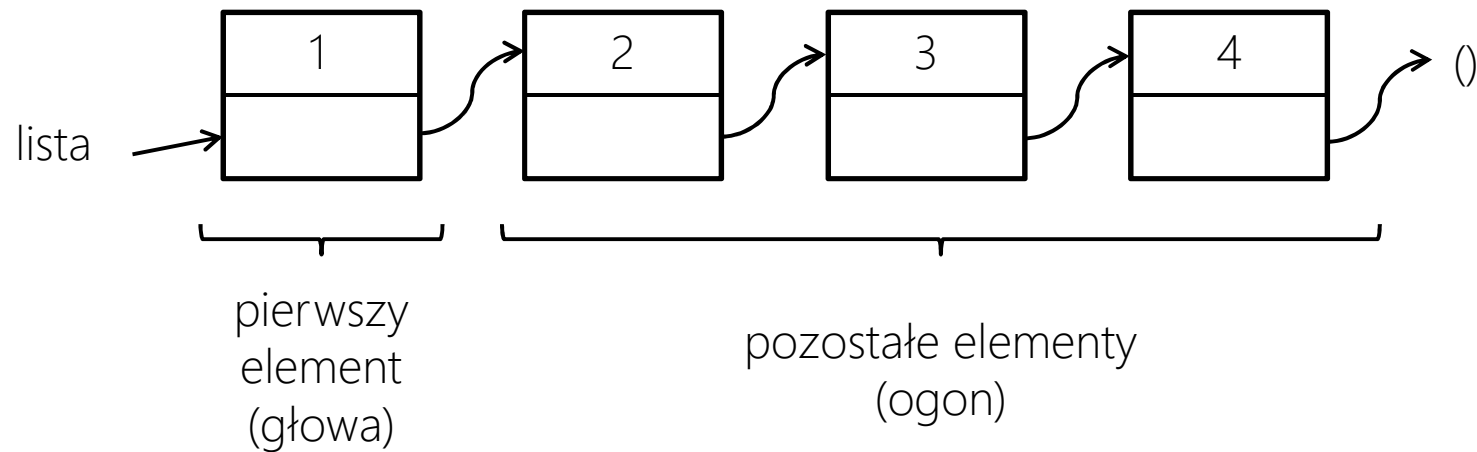
# Programowanie imperatywne

---



# Programowanie funkcyjne

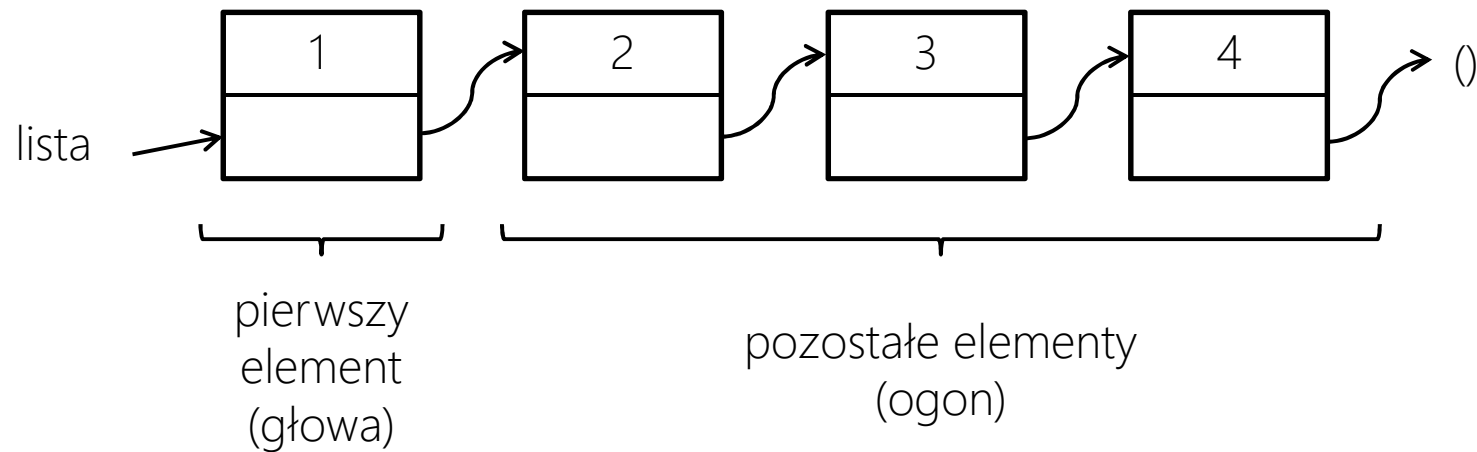
---



1. Jeżeli lista jest pusta to zwróć 0.
2. Jeżeli lista nie jest pusta to pobierz wartość pierwszego elementu na liście i dodaj go do sumy pozostałych elementów na liście

# Programowanie funkcyjne

---



```
let rec sumuj lista =  
    if List.isEmpty lista then  
        0  
    else  
        lista.Head + sumuj lista.Tail;;
```

# Programowanie funkcyjne

---

Alonzo Church  
(twórca rachunku lambda)



1903 - 1995

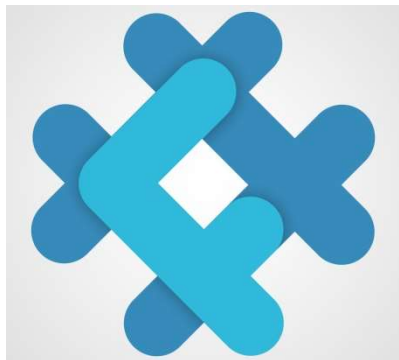
John McCarthy  
(projektant języka Lisp)



1927 - 2011

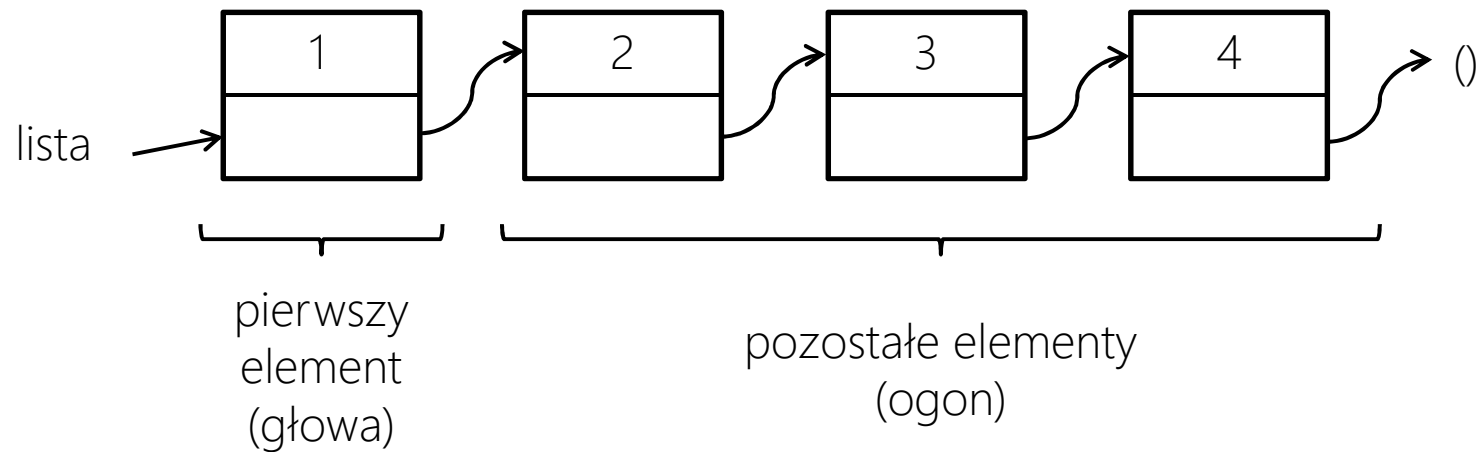
# Programowanie funkcyjne

---



# Programowanie logiczne

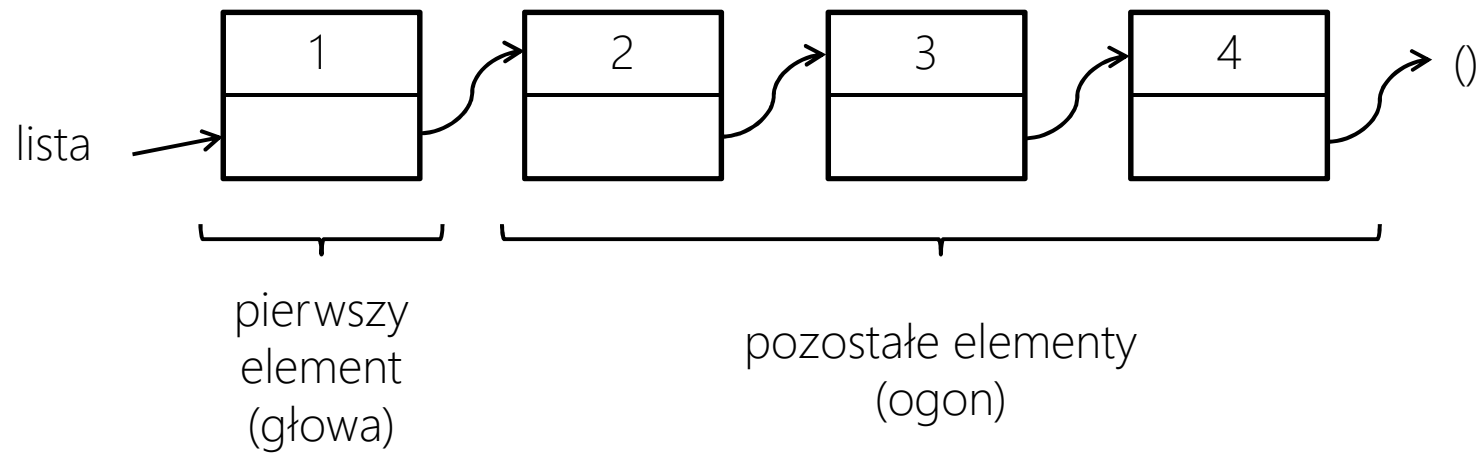
---



1. Jeżeli lista jest listą pustą to prawdą jest, że suma jej elementów  $Y$  równa się 0.
2. Jeżeli lista ma głowę  $X$  i ogon  $Xs$  oraz suma elementów w ogonie to  $Suma$ , to prawdą jest, że odpowiedzią jest  $X$  plus  $Suma$

# Programowanie logiczne

---

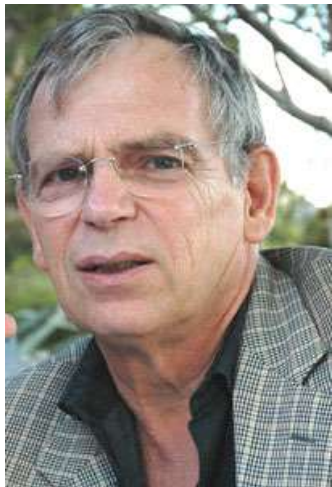


```
sumuj([], Y) :- Y is 0.  
sumuj([X|Xs], Y) :- sumuj(Xs, Suma), Y is Suma+X.
```

# Programowanie logiczne

---

Alain Colmerauer  
Twórca języka Prolog



1941-

Robert "Bob" Anthony Kowalski  
Twórca proceduralnej interpretacji  
klauzul Horna



1941-



# Programowanie logiczne

---

 **ProLog**

 **datalog.ai**<sup>®</sup>



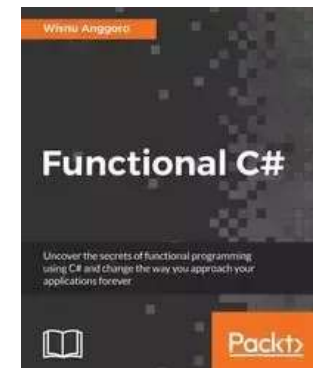
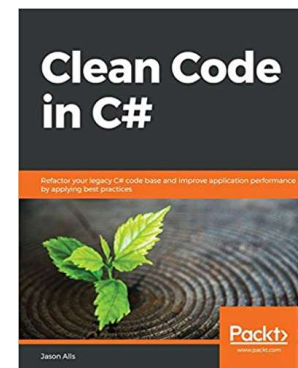
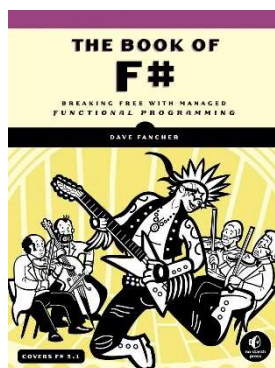
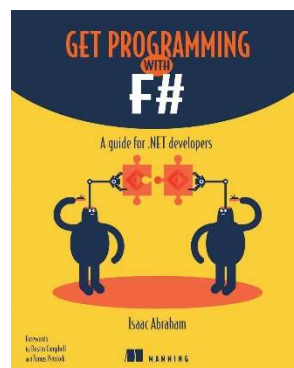
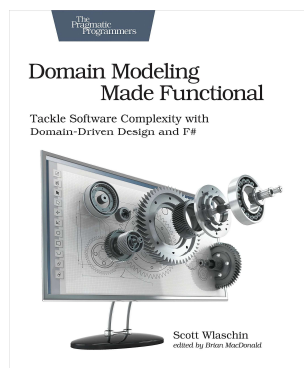
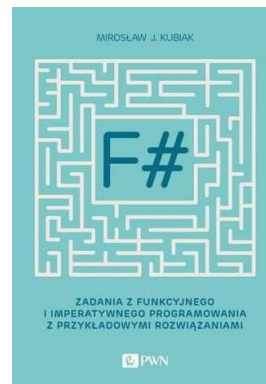
Zaliczenie przedmiotu

---

Przedmiot kończy się  
zaliczeniem

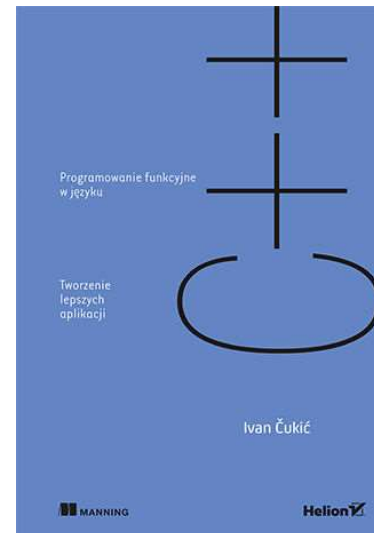
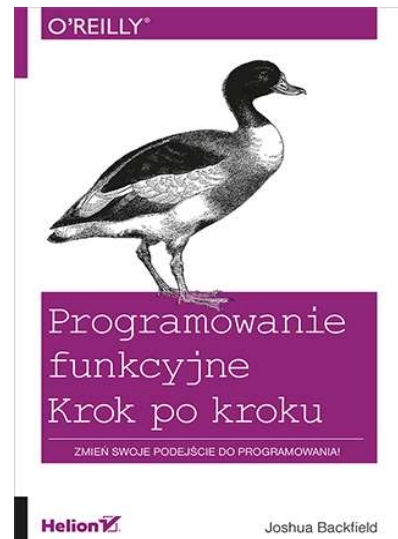
# Literatura

---



# Literatura

---



## Materiały w internecie

---

- Coursera
- Pluralsight

## Materiały z przedmiotu

---

<https://1drv.ms/u/s%21AiMGnlQluUxNdrPtpFD4pHWdRak?e=019dO2>

<http://tiny.cc/pcz-paradygmaty>