Шаблони(patterns) и правила. Правила за трансформиране.

Правила

Правилата дават възможност да се трансформира един израз от една форма в друга. Има хиляди вградени правила, но може да се създават и потребителски.
Използването на правилата нормално се свързват с процедурното и функционалното програмиране.

Когато дефинираме функция f ние дефинираме правило, което казва, че f има аргумент и всеки път, когато на този аргумент се даде стойност, то тази стойност се замества в израза. Това правило се прилага автоматично когато се запише f[B], където B може да е число, може да израз, може да е списък и т.н

```
In[1]:= f[x_] := x^2

In[2]:= f[5 i]
Out[2]= -25
```

Правила-продължение

Правило може да се създаде и чрез прилагане на оператор за заместване /.
Това правило, пък може да се използва за трансформиране на един израз в друг.
Например, правилото да се събират съответните елементи в наредените двойки.

```
In[5]:= {{x1, y1}, {x2, y2}, {x3, y3}} /. {x_, y_} :> x + y
Out[5]= {x1 + y1, x2 + y2, x3 + y3}
```

Шаблони

Шаблоните представят цели класове от изрази.

Шаблон е израз, който съдържа долна черта:

единична (_),
двойна (__), или
тройна(___).

Има различни видове шаблони в *Mathematica*

1. Шаблони с една долна черта

Това са най-простите шаблони.
Използват се, когато се дефинира потребителска функция.

```
In[1]:= f[x_] := x + 1
```

Лявата страна на горния израз е шаблон. Той съдържа шаблонен обект, *долната черта*, който може да стои след всеки израз, не само след буквата x, но е за препоръчване да е име на променлива

```
In[2]:= f[ξ]
Out[2]= 1 + ξ

In[3]:= f[Bob]
Out[3]= 1 + Bob
```

2. Шаблони, с главна част

Понякога, е необходимо да разглеждаме по-тесен кръг от шаблон. Например, може да искаме да дефинираме функция, на която аргументите приемат *само цели* стойности, или функция, чийто аргументи *са само изображения*.
Използваме шаблон с главна част от вида *_head*.

Шаблоните с главна част са особено полезни, за да ограничим вида на аргумента при дефиниране на потребителска функция. Например, можем да използваме шаблон за дефиниране на функция, чийто аргумент е САМО цяло число. Тогава използваме шаблон с главна част *_Integer*

```
In[16]:= f[x_Integer] := x + 1
```

Тогава ако пишем като аргумент цялото число 5, то се изчислява стойността на функцията

```
In[17]:= f[5]
Out[17]= 6
```

Но, ако пишем дробно число, не се изчислява

```
In[18]:= f[1.25]
Out[18]= f[1.25]
```

3. Структурирани шаблони

Шаблоните могат да се състоят и от списък.

За да илюстрираме действието на шаблоните, ще използваме вградени функции за работа със шаблони. Една от тях е:
Cases [списък, шаблон] дава елементите от списъка, който отговарят на шаблона

Например, шаблонът **{p_, q_}** подбира списъци от два елемента.

```
In[20]:= Cases[{{a, b}, {}, {1, 0}, {c, d, 3}}, {p_, q_}]
Out[20]= {{a, b}, {1, 0}}
```

В действителност, не е необходимо да наименуваме елементите на шаблона, както в горния пример, т.е. може

```
In[21]:= Cases[{{a, b}, {}, {1, 0}, {c, d, 3}}, {_, _}]
Out[21]= {{a, b}, {1, 0}}
```

Някои особености

Да разгледаме следния шаблон в символичен израз, и в числителя и в знаменателя

```
In[22]:= Cases[{2, 9/3, 1/3, x/(y + z)}, a_/b_]

Out[22]= {x/(y + z)}
```

Но защо не се избира и дробта 9/3? Защото *Mathematica отначало изчислява елементите на списъка, така , че 9/3 се редуцира до цяло, наистина*

```
In[23]:= 9/3

Out[23]= 3
```

Но което е още по странно, не се избира и дробта 1/3, заради вътрешното представяне на тази дроб

Шаблонът подбира изрази, които се основават на структурата, не на това какво означава израза или до какво би се редуцирал. Това е важно условие при използването на шаблони.

Структурираните шаблони дават възможност да се създават и правила, които да се прилагат само към някои части на израз.

Например, в следната потребителска функция аргументът се състои от списък от два израза

```
In[26]:= f[{x_, y_}] := x^2/y^3
```

Тогава, ако пишем списък като аргумент

```
In[27]:= f[{a, b}]
Out[27]= a^2/b^3
```

Но шаблонът не отговаря на следния случай, когато пишем два аргумента, неоформени в списък

```
In[28]:= f[a, b]
Out[28]= f[a, b]
```

Функцията f очаква списък от два елемента, а не редица от два елемента.

И още един пример:

```
In[7]:= h[{x_, y_}] := x^y

In[10]:= h[{2, 3}]
Out[10]= 8

In[11]:= h[{4, 3}]
Out[11]= 64
```

Шаблони с двойна и с тройна долна черта

двойна долна черта извлича всеки списък който се състои от един или повече елемента

Например, шаблонът {p__} извлича всеки списък който се състои от един или повече елемента .

```
In[32]:= Cases[{{}, {a}, {b, c}, {d, e, f}}, {p__}]
Out[32]= {{a}, {b, c}, {d, e, f}}
```

тройна долна черта представя редица от изрази с нулев брой и повече елемента, т.е. този шаблон извлича всеки списък, който има 0 или повече елементи. (може да се пише име p, но може и без него)

```
In[34]:= Cases[{{}, {a}, {b, c}, {d, e, f}}, {___}]
Out[34]= {{}, {a}, {b, c}, {d, e, f}}
```

Условни шаблони

Поставяне на условие: използва се, за да замести условието в означената част на израза

Има вида ***Израз/; тест*** и се изпълнява само ако *тест* е TRUE
Може /; да се замени с ?, т.е.е има вида **израз?тест**

```
In[48]:= Cases[{1, 2, 3, 4, 5, 6, 7, 8, 9}, n_ /; IntegerQ[√n]]
Out[48]= {1, 4, 9}
```

Това е условието

Условието в шаблона е причината да се извадят само числата, чийто корен е цяло число

```
In[49]:= Cases[{x, x^2, x^3, x^4, x^5}, _^(n_) /; EvenQ[n]]
Out[49]= {x^2, x^4}
```

Съгласно условието в шаблона се вадят всички изрази, чийто степени са четни

Шаблони с алтернативи

Означават се a|b|c|d и.т.н. където a,b,c,d са шаблони

```
In[86]:= Cases[{1, 3.1, 2/3, x, 3 + 4 I, "Hello"},
           _Integer | _Rational | _Real]

Out[86]= {1, 3.1, 2/3}
```

Изважда всички числа които са *цели* или *рационални* или *реални*

Но ако искаме да извадим всички *четни* или всички , *които са точен квадрат*

```
In[24]:= Cases[{1, 2, 3, 4, 5, 6, 7, 8, 9}, n_ /;
           EvenQ[n] || IntegerQ[Sqrt[n]]]

Out[24]= {1, 2, 4, 6, 8, 9}
```

Използват се когато има 2 и повече условия с вградени функции

Използване на условни шаблони за дефиниране на функции с ограничения

```
In[92]:= Clear[f]

In[93]:= f[x_ /; NumericQ[x]] := x + 1
```

Дефинираме функция на която аргумента е само ЦЕЛИ числа

```
In[94]:= f[2]
Out[94]= 3
```

Числото 2 е цяло и затова се извършва действието и се намира стойността на функцията

```
In[95]:= f[a]
Out[95]= f[a]
```

Аргументът НЕ е цяло (не отговаря на шаблона) и затова НЕ се извършва действието и се намира стойността на функцията

```
In[96]:= f[x_ /; StringQ[x]] := x + 1
```

Дефинираме функция на която аргумента е стринг. Понеже не сме използвали Clear, то може и аргументът да е цяло число.

```
In[97]:= f[a]
Out[97]= f[a]
```

```
In[98]:= f["a"]
Out[98]= 1 + a
```

```
In[99]:= f[x_ /; EvenQ[x]] := x / 2

In[100]:= f[x_ /; OddQ[x]] := 3 x + 1
```

Какво функция е дефинирана?

функция чийто аргумент x е само цяло число, и ако е четно се съпоставя x/2, а ако е нечетно – 3x+1.

Пример. Да се дефинира функцията $f(x) = \begin{cases} \sqrt{x} & \text{if } x \geq 0 \\ \sqrt{-x} & \text{if } x < 0 \end{cases}$

и да се начертае графиката й в интервала $-1 \leq x \leq 1.$

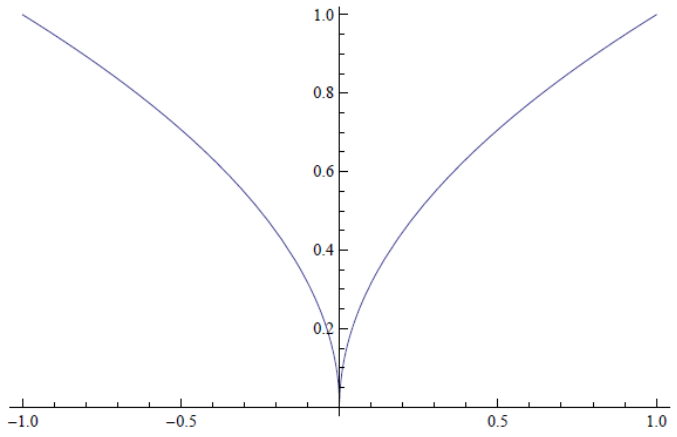
```
In[87]:= f[x_?Positive] := Sqrt[x]

In[88]:= f[x_?Negative] := Sqrt[-x]
```

дефинираме функцията

чертаем графиката й

```
In[89]:= Plot[f[x], {x, -1, 1}]
```

Out[89]=



Функции, използващи шаблони.

Cases [списък, шаблон] дава елементите от списъка, който отговарят на шаблона

DeleteCases [списък, шаблон] премахва елементите от списъка, който отговарят на шаблона

Position [списък, шаблон] дава позициятана елементите от списъка, който отговарят на шаблона

Count [списък, шаблон] брои колко елементи от списъка отговарят на шаблона

Примери за използване функциите за шаблони

```
In[2]:= Position[{f[a], g[b], f[c]}, f[x_]]
Out[2]= {{1}, {3}}
```

```
In[1]:= Cases[ {3, 4, x, x^2, x^3}, x^_ ]
Out[1]= {x^2, x^3}
```

```
In[2]:= Count[ {3, 4, x, x^2, x^3}, x^_ ]
Out[2]= 2
```

```
In[6]:= DeleteCases[{3, 4, x, x^2, x^3}, x^_]
Out[6]= {3, 4, x}
```

Задача. Създайте списък от 20 случайно избрани цели числа в интервала [2,50] и извадете всички от тях, които се делят на 3.

```
In[1]:= a = RandomInteger[{2, 50}, 20]
Out[1]= {5, 14, 10, 15, 49, 3, 49, 31, 43, 42, 48, 30, 15, 27, 2, 16, 25, 48, 8, 18}

In[2]:= Cases[a, n_ /; IntegerQ[n / 3]]
Out[2]= {15, 3, 42, 48, 30, 15, 27, 48, 18}
```

Задача. Създайте списък от 20 случайно избрани цели числа в интервала [2,50] и извадете всички от тях, които се делят на 3 или на 5.

Използваме || за да обозначим ИЛИ

```
In[8]:= Cases[a, n_ /; IntegerQ[n / 3] || IntegerQ[n / 5]]
Out[8]= {5, 10, 15, 3, 42, 48, 30, 15, 27, 25, 48, 18}
```

Може да използваме и функцията **Mod[n,k]** която дава остатъка при деление n на к

```
In[9]:= Cases[a, n_ /; Mod[n, 3] == 0 || Mod[n, 5] == 0]
Out[9]= {5, 10, 15, 3, 42, 48, 30, 15, 27, 25, 48, 18}
```

Задача. Създайте списък от 20 случайно избрани цели числа в интервала [2,50] и премахнете всички от тях, които се делят на 3 или на 5.

```
In[10]:= DeleteCases[a, n_ /; Mod[n, 3] == 0 || Mod[n, 5] == 0]
Out[10]= {14, 49, 49, 31, 43, 2, 16, 8}
```

Задача. Създайте списък от 20 случайно избрани цели числа в интервала [2,50] и пребройте колко от тях се делят на 3 или на 5.

```
In[11]:= Count[a, n_ /; Mod[n, 3] == 0 || Mod[n, 5] == 0]
Out[11]= 12
```

Задача. Създайте списък от 20 случайно избрани цели числа в интервала [2,50] и пребройте колко от тях се делят на 3 или на 5.

```
In[12]:= Position[a, n_ /; Mod[n, 3] == 0 || Mod[n, 5] == 0]
Out[12]= {{1}, {3}, {4}, {6}, {10}, {11}, {12}, {13}, {14}, {17}, {18}, {20}}
```

Правила за трансформиране

1. Създаване и използване на трансформации

Една от най честите форми на трансформациите е от вида Израз/. Правило

Всяка част в израза която съответства чрез шаблона на правилото, се записва чрез това правило.

```
In[1]:= x + y /. y → α

Out[1]= x + α
```

В примера по-долу правилото в трансформацията е шаблон, който указва, че във всяка наредена двойка елементите й се разменят.

```
In[11]:= {{3, 4}, {7, 2}, {1, 5}} /. {x_, y_} :> {y, x}
Out[11]= {{4, 3}, {2, 7}, {5, 1}}
```

Примери за използване на правило за трансформации

```
x^2 - 3 /. x -> Sqrt[3]
0
```

```
x + y /. {x -> a, y -> b}
a + b
```

```
x + 2 y /. {x -> y, y -> a}
2 a + y
```

Минава през трансформацията само веднъж.

Ако искаме повторна трансформация(да се изпълни повторно)

```
x^2 + y /. x -> y /. y -> x
x + x^2
```

Или използваме двойна наклонена черта

```
x + 2 y //. {x -> y, y -> a}
3 a
```

КРАЙ