Лекция 2

Функции и их свойства

Функциональное и логическое программирование

КамчатГТУ, 2013 г.

1. Функция в математике
   - Определение
   - Способы описания
   - Некоторые специальные функции
2. Функции в программировании
   - Синтаксис применения и определения функций
   - Функции, как объекты первого класса
   - Чистота функций
3. Базовые принципы ФП
   - Переменные и присваивание
   - Программа, данные, процесс, результат

Математическое определение

Функция — математическое понятие, отражающее связь между элементами множеств.

Математическое определение

Функция — математическое понятие, отражающее связь между элементами множеств.

Определение

Функция f **(отображение, операция)** — это **закон** или **правило**, согласно которому каждому элементу x из множества X ставится в соответствие **единственный** элемент y из множества Y.

Математическое определение

Функция — математическое понятие, отражающее связь между элементами множеств.

Определение

Функция f (отображение, операция) — это **закон** или **правило**, согласно которому каждому элементу x из множества X ставится в соответствие **единственный** элемент y из множества Y.

При этом говорят, что функция f **задана на множестве** X, или что f **отображает** X **в** Y.

$$f : X \to Y, \qquad f : x \mapsto y.$$

X — **область определения:** $X = \mathrm{dom}(f)$
Y — **область значения:** $Y = \mathrm{ran}(f)$

x — **аргумент**
y — **значение**

Способы описания

Обычно функция задаётся с помощью формулы с использованием формальных аргументов.

Способы описания

Обычно функция задаётся с помощью формулы с использованием формальных аргументов.

Примеры:

$$\mathrm{sqr}(x) = x^2;$$

$$\mathrm{mean}(x, y) = \frac{x + y}{2};$$

$$\mathrm{abs}(v) = |v|;$$

$$f(x) = \begin{cases} x^2, & x <= 0, \\ x^3, & x > 0. \end{cases}$$

Способы описания

Обычно функция задаётся с помощью формулы с использованием формальных аргументов.

Примеры:

$$\mathrm{sqr}(x) = x^2;$$

$$\mathrm{mean}(x, y) = \frac{x + y}{2};$$

$$\mathrm{abs}(v) = |v|;$$

$$f(x) = \begin{cases} x^2, & x <= 0, \\ x^3, & x > 0. \end{cases}$$

Бывает полезно определять функции, не давая им имён.

Способы описания

Обычно функция задаётся с помощью формулы с использованием формальных аргументов.

Примеры:

$$\mathrm{sqr}(x) = x^2;$$

$$\mathrm{mean}(x, y) = \frac{x + y}{2};$$

$$\mathrm{abs}(v) = |v|;$$

$$f(x) = \begin{cases} x^2, & x <= 0, \\ x^3, & x > 0. \end{cases}$$

Бывает полезно определять функции, не давая им имён.

Примеры:

$$x \mapsto x^2;$$

$$(x, y) \mapsto \frac{x + y}{2};$$

$$v \mapsto |v|;$$

$$x \mapsto \begin{cases} x^2, & x <= 0, \\ x^3, & x > 0. \end{cases}$$

Способы описания

Словесное описание

Иногда используют словесное или алгоритмическое описание функции

Способы описания

Словесное описание

Иногда используют словесное или алгоритмическое описание функции

Примеры:

Функция Дирихле I_Q равна 1 для рациональных аргументов и 0 – для иррациональных.

Способы описания

Словесное описание

Иногда используют словесное или алгоритмическое описание функции

Примеры:

Функция Дирихле I_Q равна 1 для рациональных аргументов и 0 – для иррациональных.

Функция $\mathrm{NP}(n)$ находит ближайшее число, большее n, которое не делится ни на одно простое число, меньшее $\sqrt{n}$.

Способы описания

Словесное описание

Иногда используют словесное или алгоритмическое описание функции

Примеры:

Функция Дирихле I_Q равна 1 для рациональных аргументов и 0 – для иррациональных.

Функция $\mathrm{NP}(n)$ находит ближайшее число, большее n, которое не делится ни на одно простое число, меньшее $\sqrt{n}$.

Логарифмическая функция $\log_a b$ показывает, в какую степень необходимо возвести число a, чтобы получилось число b.

Способы описания

Табулирование

Функцию можно задать, перечислив все её возможные аргументы и значения для них.

Способы описания

Табулирование

Функцию можно задать, перечислив все её возможные аргументы и значения для них.

Пример:

Определение логической операции ИЛИ

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

Способы описания

Табулирование

Функцию можно задать, перечислив все её возможные аргументы и значения для них.

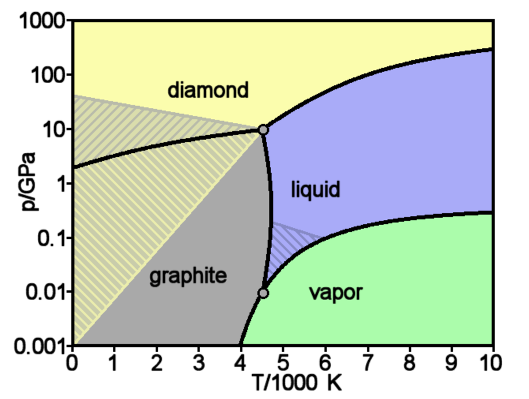
Пример:

Определение логической операции ИЛИ

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

Пример:

Фазовая диаграмма углерода



Способы описания

Рекурсивное определение

Функция может быть задана **рекурсивно**, то есть через саму себя. В этом случае одни значения функции определяются через другие её значения.

Способы описания

Рекурсивное определение

Функция может быть задана **рекурсивно**, то есть через саму себя. В этом случае одни значения функции определяются через другие её значения.

Пример:

Определение Гамма функции

$$\Gamma(x+1) = x\Gamma(x)$$
$$\Gamma(1) = 1$$

Способы описания

Рекурсивное определение

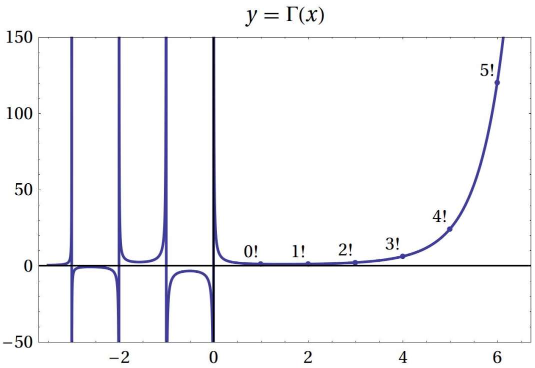
Функция может быть задана **рекурсивно**, то есть через саму себя. В этом случае одни значения функции определяются через другие её значения.

Пример:

Определение Гамма функции

$$\Gamma(x+1) = x\Gamma(x)$$
$$\Gamma(1) = 1$$

Эта функция обобщает факториал. Для $x \in Z$ $\Gamma(x) = (x-1)!$



Способы описания

Рекурсивное определение

Функция может быть задана **рекурсивно**, то есть через саму себя. В этом случае одни значения функции определяются через другие её значения.

Пример:

Определение Гамма функции

$$\Gamma(x+1) = x\Gamma(x)$$
$$\Gamma(1) = 1$$

Эта функция обобщает факториал. Для $x \in Z$ $\Gamma(x) = (x-1)!$

Пример:

Определение функции Аккермана

$$\varphi(m, n+1, p+1) = \varphi(m, \varphi(m, n, p), p)$$
$$\varphi(m, n+1, 0) = \varphi(m, n, 0) + 1$$
$$\varphi(m, 0, 1) = 0$$
$$\varphi(m, 0, 2) = 1$$
$$\varphi(m, 0, p) = m$$

Способы описания

Рекурсивное определение

Функция может быть задана **рекурсивно**, то есть через саму себя. В этом случае одни значения функции определяются через другие её значения.

Пример:

Определение Гамма функции

$$\Gamma(x+1) = x\Gamma(x)$$
$$\Gamma(1) = 1$$

Эта функция обобщает факториал. Для $x \in Z$ $\Gamma(x) = (x-1)!$

Пример:

Определение функции Аккермана

$$\varphi(m, n+1, p+1) = \varphi(m, \varphi(m, n, p), p)$$
$$\varphi(m, n+1, 0) = \varphi(m, n, 0) + 1$$
$$\varphi(m, 0, 1) = 0$$
$$\varphi(m, 0, 2) = 1$$
$$\varphi(m, 0, p) = m$$

Эта функция обобщает арифметику:

$$\varphi(m, n, 0) = m + n$$
$$\varphi(m, n, 1) = m \cdot n$$
$$\varphi(m, n, 2) = m^n$$
$$\varphi(m, n, 3) = \underbrace{m^{m^{\cdot^{\cdot^{m}}}}}_{n}$$

Способы описания

Определение через уравнение

Функция может быть задана функциональным или дифференциальным уравнением.

Способы описания

Определение через уравнение

Функция может быть задана функциональным или дифференциальным уравнением.

Примеры:

Определение функции $y = e^x$, как решения уравнения:

$$y'(x) = y(x),$$
$$y(0) = 1.$$

Способы описания

Определение через уравнение

Функция может быть задана функциональным или дифференциальным уравнением.

Примеры:

Определение функции $y = e^x$, как решения уравнения:

$$y'(x) = y(x),$$
$$y(0) = 1.$$

Уравнение, определяющее показательную функцию $f(x) = a^x$:

$$f(x + y) = f(x)f(y).$$

Способы описания

Определение через уравнение

Функция может быть задана функциональным или дифференциальным уравнением.

Примеры:

Определение функции $y = e^x$, как решения уравнения:

$$y'(x) = y(x),$$
$$y(0) = 1.$$

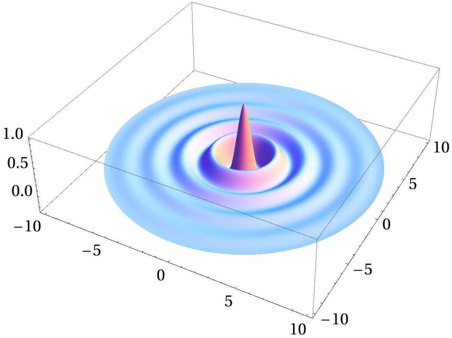
Уравнение, определяющее показательную функцию $f(x) = a^x$:

$$f(x + y) = f(x)f(y).$$

Пример:

Уравнение, определяющее функцию Бесселя:

$$x^2 F''(x) + xF'(x) - (x^2 - \alpha^2)F(x) = 0$$



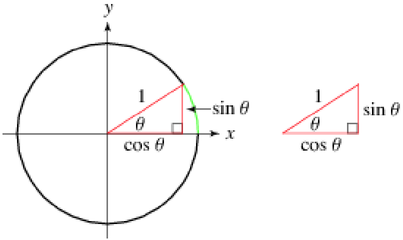
Способы описания

Определения функции синус:

Способы описания

Определения функции синус:

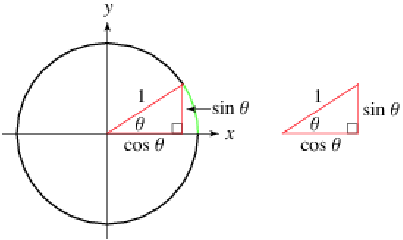
- Словесное описание: функция синус для заданного угла ϕ возвращает половину длины хорды единичной окружности, стянутой углом 2ϕ.



Способы описания

Определения функции синус:

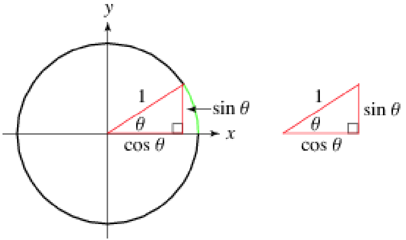
- Словесное описание: функция синус для заданного угла ϕ возвращает половину длины хорды единичной окружности, стянутой углом 2ϕ.
- С помощью формулы: $\sin(x) = \sum_{i=1}^{\infty} (-1)^{2i-1} \frac{x^i}{i!}$

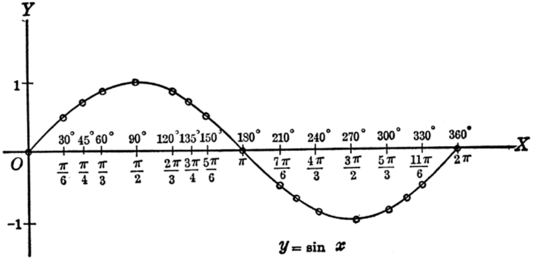


Способы описания

Определения функции синус:

- Словесное описание: функция синус для заданного угла ϕ возвращает половину длины хорды единичной окружности, стянутой углом 2ϕ.
- С помощью формулы: $\sin(x) = \sum_{i=1}^{\infty} (-1)^{2i-1} \frac{x^i}{i!}$
- С помощью таблицы или графика.





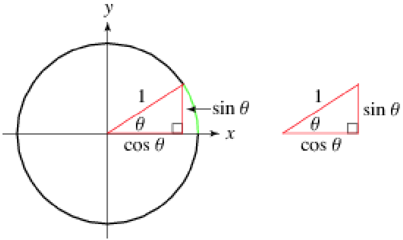
Способы описания

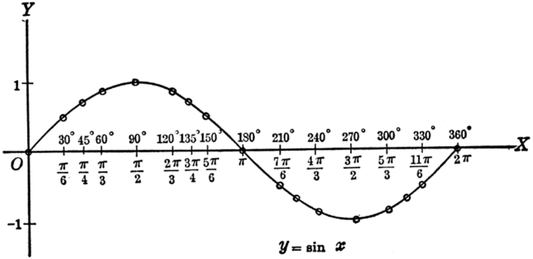
Определения функции синус:

- Словесное описание: функция синус для заданного угла ϕ возвращает половину длины хорды единичной окружности, стянутой углом 2ϕ.
- С помощью формулы: $\sin(x) = \sum\limits_{i=1}^{\infty} (-1)^{2i-1} \frac{x^i}{i!}$
- С помощью таблицы или графика.
- Рекурсивное:

$$\sin x = 3 \sin \frac{x}{3} - 4 \sin^3 \frac{x}{3}$$

$$\sin x = x, \quad \text{при} \quad x \to 0$$





Способы описания

Определения функции синус:

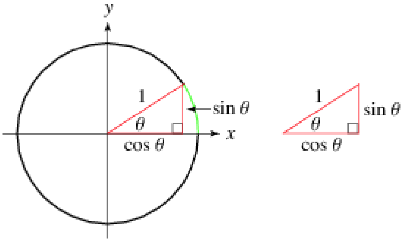
- Словесное описание: функция синус для заданного угла ϕ возвращает половину длины хорды единичной окружности, стянутой углом 2ϕ.
- С помощью формулы: $\sin(x) = \sum_{i=1}^{\infty} (-1)^{2i-1} \frac{x^i}{i!}$
- С помощью таблицы или графика.
- Рекурсивное:

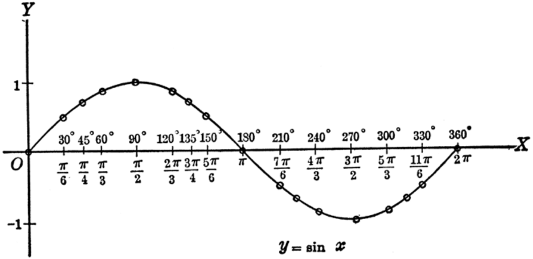
$$\sin x = 3\sin\frac{x}{3} - 4\sin^3\frac{x}{3}$$

$$\sin x = x, \quad \text{при} \quad x \to 0$$

- Через уравнение:

$$y''(x) = -y(x); \quad y'(0) = 1, \quad y(0) = 0.$$





Тождественное отображение

Определение

Функцию $\mathrm{id} : X \to X$ такую, что $\mathrm{id}(x) = x$ для всех $x \in X$, называют **тождественной функцией**, отображающей множество X в себя.

Тождественное отображение

Определение

Функцию $\mathrm{id} : X \to X$ такую, что $\mathrm{id}(x) = x$ для всех $x \in X$, называют **тождественной функцией**, отображающей множество X в себя.

Тождественная функция выполняет роль единичного элемента во многих функциональных преобразованиях.

Тождественное отображение

Определение

Функцию $\mathrm{id} : X \to X$ такую, что $\mathrm{id}(x) = x$ для всех $x \in X$, называют **тождественной функцией**, отображающей множество X в себя.

Тождественная функция выполняет роль единичного элемента во многих функциональных преобразованиях.

Примеры:

$$\mathrm{id} = x \mapsto x$$

Тождественное отображение

Определение

Функцию $\mathrm{id} : X \to X$ такую, что $\mathrm{id}(x) = x$ для всех $x \in X$, называют **тождественной функцией**, отображающей множество X в себя.

Тождественная функция выполняет роль единичного элемента во многих функциональных преобразованиях.

Примеры:

$$\mathrm{id} = x \mapsto x$$

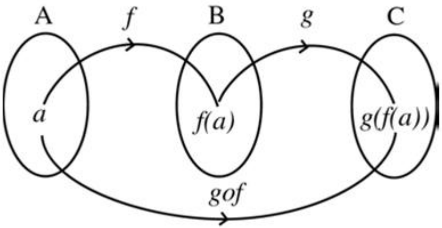
$$\mathrm{id}(\sin) = \sin$$

Композиция функций

Пусть $f : A \to B$ и $g : B \to C$ — отображения такие, что $\mathrm{ran}(f) \subset \mathrm{dom}(g)$, тогда для каждого $a \in A$ однозначно определён $c \in C$, а значит существует функция $h : A \to C$ такая, что

$$h(x) = g(f(x)).$$

Это отображение называется **композицией функций** f **и** g и обозначается $h = g \circ f$.

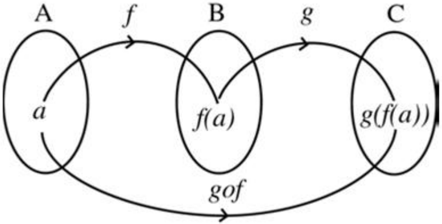


Композиция функций

Пусть $f : A \to B$ и $g : B \to C$ — отображения такие, что $\mathrm{ran}(f) \subset \mathrm{dom}(g)$, тогда для каждого $a \in A$ однозначно определён $c \in C$, а значит существует функция $h : A \to C$ такая, что

$$h(x) = g(f(x)).$$

Это отображение называется **композицией функций** f **и** g и обозначается $h = g \circ f$.



Пример:

Функцию $f(x) = \sin^2(x)$ можно представить, как композицию:

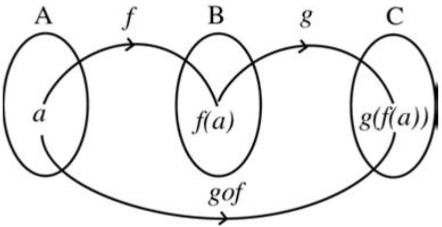
$$f = (x \mapsto x^2) \circ \sin$$

Композиция функций

Пусть $f : A \to B$ и $g : B \to C$ — отображения такие, что $\operatorname{ran}(f) \subset \operatorname{dom}(g)$, тогда для каждого $a \in A$ однозначно определён $c \in C$, а значит существует функция $h : A \to C$ такая, что

$$h(x) = g(f(x)).$$

Это отображение называется **композицией функций** f **и** g и обозначается $h = g \circ f$.



Пример:

Функцию $f(x) = \sin^2(x)$ можно представить, как композицию:

$$f = (x \mapsto x^2) \circ \sin$$

а функцию $\cos(x)$ таким образом:

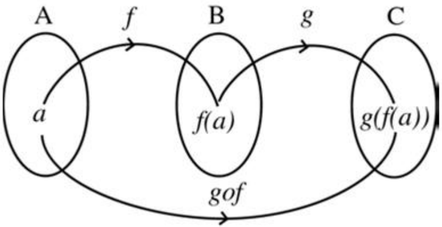
$$\cos = (x \mapsto \sqrt{1 - x}) \circ (x \mapsto x^2) \circ \sin,$$

Композиция функций

Пусть $f : A \to B$ и $g : B \to C$ — отображения такие, что $\mathrm{ran}(f) \subset \mathrm{dom}(g)$, тогда для каждого $a \in A$ однозначно определён $c \in C$, а значит существует функция $h : A \to C$ такая, что

$$h(x) = g(f(x)).$$

Это отображение называется **композицией функций** f **и** g и обозначается $h = g \circ f$.



Пример:

Функцию $f(x) = \sin^2(x)$ можно представить, как композицию:

$$f = (x \mapsto x^2) \circ \sin$$

а функцию $\cos(x)$ таким образом:

$$\cos = (x \mapsto \sqrt{1-x}) \circ (x \mapsto x^2) \circ \sin,$$

или таким:

$$\cos = \sin \circ (x \mapsto \pi/2 - x).$$

Функции высших порядков

Определение

Функции, область определения или область значения которых включают в себя множество функций, называются **функциями высших порядков (операторами)**.

Функции высших порядков

Определение

Функции, область определения или область значения которых включают в себя множество функций, называются **функциями высших порядков (операторами)**.

Пример:

Операторы дифференцирования и интегрирования:

$$\frac{d}{dx}\sin(x) = \cos(x), \quad \mathrm{D}(\sin) = \cos$$

Функции высших порядков

Определение

Функции, область определения или область значения которых включают в себя множество функций, называются **функциями высших порядков (операторами)**.

Пример:

Операторы дифференцирования и интегрирования:

$$\frac{d}{dx}\sin(x) = \cos(x), \quad \mathrm{D}(\sin) = \cos$$

$$\int \sin(x)dx = -\cos(x), \quad \mathrm{I}(\sin) = (x \mapsto -x) \circ \cos$$

Функции высших порядков

Определение

Функции, область определения или область значения которых включают в себя множество функций, называются **функциями высших порядков (операторами)**.

Пример:

Операторы дифференцирования и интегрирования:

$$\frac{d}{dx}\sin(x) = \cos(x), \quad \mathrm{D}(\sin) = \cos$$

$$\int \sin(x)dx = -\cos(x), \quad \mathrm{I}(\sin) = (x \mapsto -x) \circ \cos$$

оператор удвоения аппликации:

$$\mathrm{dup}(f) = f \circ f$$

Функции высших порядков

Определение

Функции, область определения или область значения которых включают в себя множество функций, называются **функциями высших порядков (операторами)**.

Пример:

Операторы дифференцирования и интегрирования:

$$\frac{d}{dx}\sin(x) = \cos(x), \quad \mathrm{D}(\sin) = \cos$$

$$\int \sin(x)dx = -\cos(x), \quad \mathrm{I}(\sin) = (x \mapsto -x) \circ \cos$$

оператор удвоения аппликации:

$$\mathrm{dup}(f) = f \circ f$$

$$\mathrm{dup}(x \mapsto x^2) =$$

Функции высших порядков

Определение

Функции, область определения или область значения которых включают в себя множество функций, называются **функциями высших порядков (операторами)**.

Пример:

Операторы дифференцирования и интегрирования:

$$\frac{d}{dx}\sin(x) = \cos(x), \quad \mathrm{D}(\sin) = \cos$$

$$\int \sin(x)dx = -\cos(x), \quad \mathrm{I}(\sin) = (x \mapsto -x) \circ \cos$$

оператор удвоения аппликации:

$$\mathrm{dup}(f) = f \circ f$$

$$\mathrm{dup}(x \mapsto x^2) = x \mapsto x^4$$

Обратная функция

Определение

Функция $g : Y \to X$ называется обратной функции $f : X \to Y$ если

$$g \circ f = \mathrm{id},$$
$$f \circ g = \mathrm{id}.$$

Обратная функция

Определение

Функция $g : Y \to X$ называется обратной функции $f : X \to Y$ если

$$g \circ f = \mathrm{id},$$
$$f \circ g = \mathrm{id}.$$

Пример:

$$\mathrm{inv}(\ln) =$$

Обратная функция

Определение

Функция $g : Y \to X$ называется обратной функции $f : X \to Y$ если

$$g \circ f = \mathrm{id},$$
$$f \circ g = \mathrm{id}.$$

Пример:

$$\mathrm{inv}(\ln) = \exp$$

Обратная функция

Определение

Функция $g : Y \to X$ называется обратной функции $f : X \to Y$ если

$$g \circ f = \mathrm{id},$$
$$f \circ g = \mathrm{id}.$$

Пример:

$$\mathrm{inv}(\ln) = \exp$$
$$\mathrm{inv}(\sin) =$$

Обратная функция

Определение

Функция $g : Y \to X$ называется обратной функции $f : X \to Y$ если

$$g \circ f = \mathrm{id},$$
$$f \circ g = \mathrm{id}.$$

Пример:

$$\mathrm{inv}(\ln) = \exp$$
$$\mathrm{inv}(\sin) = \arcsin$$

Обратная функция

Определение

Функция $g : Y \to X$ называется обратной функции $f : X \to Y$ если

$$g \circ f = \mathrm{id},$$
$$f \circ g = \mathrm{id}.$$

Пример:

$$\mathrm{inv}(\ln) = \exp$$
$$\mathrm{inv}(\sin) = \arcsin$$
$$\mathrm{inv}(x \mapsto \sqrt{x}) =$$

Обратная функция

Определение

Функция $g : Y \to X$ называется обратной функции $f : X \to Y$ если

$$g \circ f = \mathrm{id},$$
$$f \circ g = \mathrm{id}.$$

Пример:

$$\mathrm{inv}(\ln) = \exp$$
$$\mathrm{inv}(\sin) = \arcsin$$
$$\mathrm{inv}(x \mapsto \sqrt{x}) = (x \mapsto x^2)$$

Неподвижная точка функции

Определение

Элемент x^* множества определения функции f называется **неподвижной точкой** функции f, если $f(x^*) = x^*$

Неподвижная точка функции

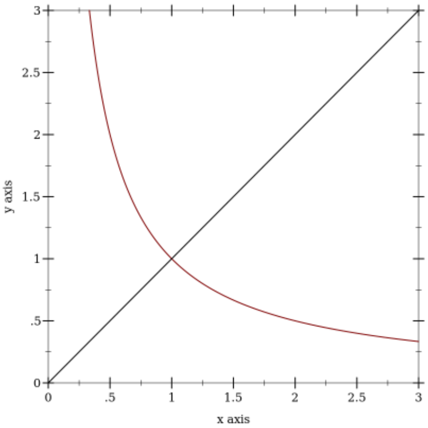
Определение

Элемент x^* множества определения функции f называется **неподвижной точкой** функции f, если $f(x^*) = x^*$

Примеры:

Функция $x \mapsto 1/x$. — число 1.

$1/1 = 1$



Неподвижная точка функции

Определение

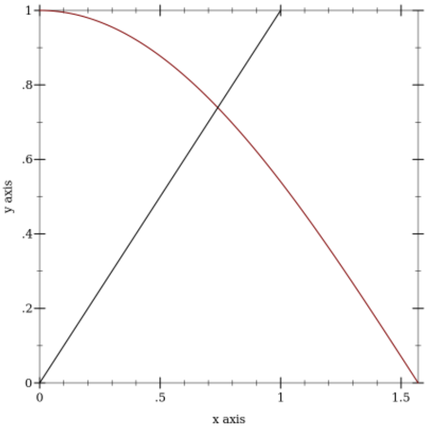
Элемент x^* множества определения функции f называется **неподвижной точкой** функции f, если $f(x^*) = x^*$

Примеры:

Функция $x \mapsto 1/x$. — число 1.

Функция $\cos$ — число 0.7391.

$cos(0.7391) = 0.7391$



Неподвижная точка функции

Определение

Элемент x^* множества определения функции f называется **неподвижной точкой** функции f, если $f(x^*) = x^*$

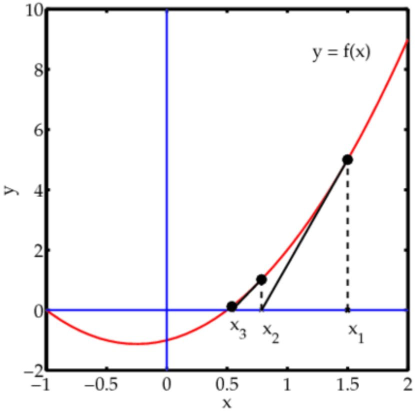
Примеры:

Функция $x \mapsto 1/x$. — число 1.

Функция $\cos$ — число 0.7391.

Преобразование Ньютона: $N_f = x \mapsto x - f(x)/f'(x)$ — корень уравнения $f(x) = 0$.

$$f(x) = 0 \implies N_f = id$$



Неподвижная точка функции

Определение

Элемент x^* множества определения функции f называется **неподвижной точкой** функции f, если $f(x^*) = x^*$

Примеры:

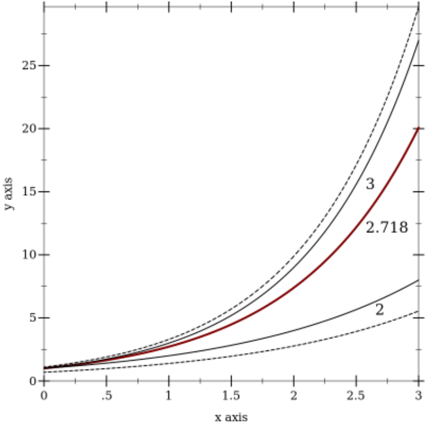
Функция $x \mapsto 1/x$. — число 1.

Функция $\cos$ — число 0.7391.

Преобразование Ньютона: $N_f = x \mapsto x - f(x)/f'(x)$ — корень уравнения $f(x) = 0$.

Оператор производной — функция $\exp$

$$\frac{d}{dx}e^x = e^x$$



Неподвижная точка функции

Оператор неподвижной точки

Устойчивые неподвижные точки можно получить с помощью итераций, начиная с некоторой окрестности x^*:

$$f(f(\ldots f(x))) \to x^*.$$

Оператор

$$\mathrm{Y}(f) = f \circ f \circ \ldots \circ f = f(\mathrm{Y}(f))$$

называется **оператором неподвижной точки.**

Неподвижная точка функции

Оператор неподвижной точки

Устойчивые неподвижные точки можно получить с помощью итераций, начиная с некоторой окрестности x^*:

$$f(f(\dots f(x))) \to x^*.$$

Оператор

$$\mathrm{Y}(f) = f \circ f \circ \dots \circ f = f(\mathrm{Y}(f))$$

называется **оператором неподвижной точки.**

$$\begin{aligned} cos(1) &= 0.54030 \\ cos(0.54030) &= 0.85755 \\ cos(0.85755) &= 0.65429 \\ cos(0.65429) &= 0.79348 \\ cos(0.79348) &= 0.70137 \\ cos(0.70137) &= 0.76395 \\ cos(0.76395) &= 0.72211 \\ &\vdots \\ cos(0.73908) &= 0.73908 \end{aligned}$$

Закрепление материала

- Какая функция определяется функциональным уравнением

$$af(x) + bf(y) = f(x^a y^b)?$$

Закрепление материала

- Какая функция определяется функциональным уравнением

$$af(x) + bf(y) = f(x^a y^b)?$$

- Дайте словесное определение оператору производной D.

Закрепление материала

- Какая функция определяется функциональным уравнением

$$af(x) + bf(y) = f(x^a y^b)?$$

- Дайте словесное определение оператору производной D.
- Дайте алгоритмическое определение оператору inv.

Закрепление материала

- Какая функция определяется функциональным уравнением

$$af(x) + bf(y) = f(x^a y^b)?$$

- Дайте словесное определение оператору производной D.
- Дайте алгоритмическое определение оператору inv.
- Чему равно $\mathrm{dup}(x \mapsto 1/x)$?

Закрепление материала

- Какая функция определяется функциональным уравнением

$$af(x) + bf(y) = f(x^a y^b)?$$

- Дайте словесное определение оператору производной D.
- Дайте алгоритмическое определение оператору inv.
- Чему равно $\mathrm{dup}(x \mapsto 1/x)$?
- Чему равно $\mathrm{inv} \circ \mathrm{inv}$?

Закрепление материала

- Какая функция определяется функциональным уравнением

$$af(x) + bf(y) = f(x^a y^b)?$$

- Дайте словесное определение оператору производной D.
- Дайте алгоритмическое определение оператору inv.
- Чему равно $\mathrm{dup}(x \mapsto 1/x)$?
- Чему равно $\mathrm{inv} \circ \mathrm{inv}$?
- Что является неподвижной точкой следующих функций и операторов

Закрепление материала

- Какая функция определяется функциональным уравнением

$$af(x) + bf(y) = f(x^a y^b)?$$

- Дайте словесное определение оператору производной D.
- Дайте алгоритмическое определение оператору inv.
- Чему равно $\mathrm{dup}(x \mapsto 1/x)$?
- Чему равно $\mathrm{inv} \circ \mathrm{inv}$?
- Что является неподвижной точкой следующих функций и операторов
  - $\sin$, $x \mapsto \sqrt{x}$, $x \mapsto 1 + \frac{1}{x}$, $x \mapsto \sqrt{1+x}$

Закрепление материала

- Какая функция определяется функциональным уравнением

$$af(x) + bf(y) = f(x^a y^b)?$$

- Дайте словесное определение оператору производной D.
- Дайте алгоритмическое определение оператору inv.
- Чему равно $\mathrm{dup}(x \mapsto 1/x)$?
- Чему равно $\mathrm{inv} \circ \mathrm{inv}$?
- Что является неподвижной точкой следующих функций и операторов
  - $\sin$, $x \mapsto \sqrt{x}$, $x \mapsto 1 + \frac{1}{x}$, $x \mapsto \sqrt{1+x}$
  - inv, dup

Закрепление материала

- Какая функция определяется функциональным уравнением

$$af(x) + bf(y) = f(x^a y^b)?$$

- Дайте словесное определение оператору производной D.
- Дайте алгоритмическое определение оператору inv.
- Чему равно $\mathrm{dup}(x \mapsto 1/x)$?
- Чему равно $\mathrm{inv} \circ \mathrm{inv}$?
- Что является неподвижной точкой следующих функций и операторов
  - $\sin$, $x \mapsto \sqrt{x}$, $x \mapsto 1 + \frac{1}{x}$, $x \mapsto \sqrt{1+x}$
  - inv, dup
  - $(x \mapsto -x) \circ \mathrm{D} \circ \mathrm{D}$

Закрепление материала

- Какая функция определяется функциональным уравнением

$$af(x) + bf(y) = f(x^a y^b)?$$

- Дайте словесное определение оператору производной D.
- Дайте алгоритмическое определение оператору inv.
- Чему равно $\mathrm{dup}(x \mapsto 1/x)$?
- Чему равно $\mathrm{inv} \circ \mathrm{inv}$?
- Что является неподвижной точкой следующих функций и операторов
    - $\sin$, $x \mapsto \sqrt{x}$, $x \mapsto 1 + \frac{1}{x}$, $x \mapsto \sqrt{1+x}$
    - inv, dup
    - $(x \mapsto -x) \circ \mathrm{D} \circ \mathrm{D}$
- Иногда вводят оператор композиции C_f, такой, что $C_f(g) = g \circ f$. Что является неподвижной точкой оператора C_{id}?

Функции в программировании

Определение

Функция — это поименованная часть программы, которая может вызываться из других частей программы столько раз, сколько необходимо.
Функция, в отличие от **процедуры**, обязательно возвращает значение.

Функции в программировании

Определение

Функция — это поименованная часть программы, которая может вызываться из других частей программы столько раз, сколько необходимо.
Функция, в отличие от **процедуры**, обязательно возвращает значение.

Тело функции

Тело функции описывает способ получения результата или определяет, что является результатом.

Функции в программировании

Определение

Функция — это поименованная часть программы, которая может вызываться из других частей программы столько раз, сколько необходимо.
Функция, в отличие от **процедуры**, обязательно возвращает значение.

Тело функции

Тело функции описывает способ получения результата или определяет, что является результатом.

Формальные аргументы

При определении функции в её теле указываются **формальные аргументы**, определяющие результат.

Функции в программировании

Определение

Функция — это поименованная часть программы, которая может вызываться из других частей программы столько раз, сколько необходимо.
Функция, в отличие от **процедуры**, обязательно возвращает значение.

Тело функции

Тело функции описывает способ получения результата или определяет, что является результатом.

Формальные аргументы

При определении функции в её теле указываются **формальные аргументы**, определяющие результат.

Фактические аргументы

При **вызове функции**, ей передаются управление вычислительным процессом и **фактические аргументы**.

Синтаксис применения функций

синтаксис **использование**

примеры

Синтаксис применения функций

синтаксис	использование	примеры			
скобочный	Fortran, C, Pascal, Erlang,	$f(x)$	$g(x, y)$	$f(g(x), y)$	$f(x, g(y))$

Синтаксис применения функций

синтаксис	использование	примеры			
скобочный	Fortran, C, Pascal, Erlang,	$f(x)$	$g(x,y)$	$f(g(x),y)$	$f(x,g(y))$
префиксный бесскобочный	Haskell, ML, F#	$f\ x$	$g\ x\ y$	$f\ (g\ x)\ y$	$f\ x\ (g\ y)$

Синтаксис применения функций

синтаксис	использование	примеры			
скобочный	FORTRAN, C, PASCAL, ERLANG,	$f(x)$	$g(x,y)$	$f(g(x),y)$	$f(x,g(y))$
префиксный бесскобочный	HASKELL, ML, F#	$f\ x$	$g\ x\ y$	$f\ (g\ x)\ y$	$f\ x\ (g\ y)$
префиксный скобочный	LISP, SCHEME, CLOJURE	$(f\ x)$	$(g\ x\ y)$	$(f\ (g\ x)\ y)$	$(f\ x\ (g\ y))$

Синтаксис применения функций

синтаксис	использование	примеры			
скобочный	Fortran, C, Pascal, Erlang,	$f(x)$	$g(x,y)$	$f(g(x),y)$	$f(x,g(y))$
префиксный бесскобочный	Haskell, ML, F#	$f\ x$	$g\ x\ y$	$f\ (g\ x)\ y$	$f\ x\ (g\ y)$
префиксный скобочный	Lisp, Scheme, Clojure	$(f\ x)$	$(g\ x\ y)$	$(f\ (g\ x)\ y)$	$(f\ x\ (g\ y))$
инфиксный	Haskell, Mathematica Scheme	$f\ x$	$x \sim g \sim y$	$(g\ x) \sim f \sim y$	$x \sim f \sim (g\ y)$

Синтаксис применения функций

синтаксис	использование	примеры			
скобочный	Fortran, C, Pascal, Erlang,	$f(x)$	$g(x,y)$	$f(g(x),y)$	$f(x,g(y))$
префиксный бесскобочный	Haskell, ML, F#	$f\ x$	$g\ x\ y$	$f\ (g\ x)\ y$	$f\ x\ (g\ y)$
префиксный скобочный	Lisp, Scheme, Clojure	$(f\ x)$	$(g\ x\ y)$	$(f\ (g\ x)\ y)$	$(f\ x\ (g\ y))$
инфиксный	Haskell, Mathematica Scheme	$f\ x$	$x \sim g \sim y$	$(g\ x) \sim f \sim y$	$x \sim f \sim (g\ y)$
постфиксный	Forth, PostScript, Joy	$x\ f$	$x\ y\ g$	$x\ g\ y\ f$	$x\ y\ g\ f$

Синтаксис применения функций

Пример:

Скобочный:

$$f(h(x, g(f(x, y), h(y, z, f(2, x))), z), g(y, z))$$

Синтаксис применения функций

Пример:

Скобочный:

$$f(h(x, g(f(x, y), h(y, z, f(2, x))), z), g(y, z))$$

Префиксный:

$$f\ (h\ x\ (g\ (f\ x\ y)\ (h\ y\ z\ (f\ 2\ x)))\ z)\ (g\ y\ z)$$

Синтаксис применения функций

Пример:

Скобочный:

$$f(h(x, g(f(x, y), h(y, z, f(2, x))), z), g(y, z))$$

Префиксный:

$$f\ (h\ x\ (g\ (f\ x\ y)\ (h\ y\ z\ (f\ 2\ x)))\ z)\ (g\ y\ z)$$

Префиксный скобочный:

```
(f (h x
        (g (f x y)
           (h y z (f 2 x)))
       z)
   (g y z))
```

Синтаксис применения функций

Пример:

Скобочный:

$$f(h(x, g(f(x, y), h(y, z, f(2, x))), z), g(y, z))$$

Префиксный:

$$f\ (h\ x\ (g\ (f\ x\ y)\ (h\ y\ z\ (f\ 2\ x)))\ z)\ (g\ y\ z)$$

Префиксный скобочный:

```
(f (h x
      (g (f x y)
         (h y z (f 2 x)))
      z)
   (g y z))
```

Постфиксный:

$$x\ x\ y\ f\ y\ z\ 2\ x\ f\ h\ g\ z\ h\ y\ z\ g\ f$$

Синтаксис определения функций

скобочный (Erlang):

```
sqr(x) -> x * x

norm(x, y) -> sqrt(sqr(x) + sqr(y))
```

Синтаксис определения функций

скобочный (Erlang):

```
sqr(x) -> x * x

norm(x, y) -> sqrt(sqr(x) + sqr(y))
```

префиксный скобочный (Scheme):

```
(define (sqr x) (* x x))

(define (norm x y)
  (sqrt (+ (sqr x)
           (sqr y))))
```

Синтаксис определения функций

скобочный (Erlang):

```
sqr(x) -> x * x

norm(x, y) -> sqrt(sqr(x) + sqr(y))
```

префиксный (Haskell):

```
sqr x = x * x

norm x y = sqrt & (sqr x) + (sqr y)
```

префиксный скобочный (Scheme):

```
(define (sqr x) (* x x)

(define (norm x y)
  (sqrt (+ (sqr x)
           (sqr y))))
```

Синтаксис определения функций

скобочный (Erlang):

```
sqr(x) -> x * x

norm(x, y) -> sqrt(sqr(x) + sqr(y))
```

префиксный (Haskell):

```
sqr x = x * x

norm x y = sqrt & (sqr x) + (sqr y)
```

префиксный скобочный (Scheme):

```
(define (sqr x) (* x x)

(define (norm x y)
  (sqrt (+ (sqr x)
           (sqr y))))
```

постфиксный (Joy):

```
DEFINE sqr == dup *

DEFINE norm == sqr swap sqr + sqrt
```

Что делает ЯП функциональными?

Функции можно определять практически во всех модульных ЯП:

C++:

```
int sqr(int x)
{
  return x * x;
}
```

Pascal:

```
function sqr(x: integer): integer;
begin
  sqr := x * x
end;
```

Что делает ЯП функциональными?

Функции можно определять практически во всех модульных ЯП:

C++:

```
int sqr(int x)
{
  return x * x;
}
```

Pascal:

```
function sqr(x: integer): integer;
begin
  sqr := x * x
end;
```

Определение

Функциональным будем называть язык программирования, в котором функции являются **объектами первого класса**.

Объект первого класса

Определение

Объект называют **объектом первого класса** в контексте конкретного ЯП, когда он:

Объект первого класса

Определение

Объект называют **объектом первого класса** в контексте конкретного ЯП, когда он:

- может быть связан с идентификатором;

Объект первого класса

Определение

Объект называют **объектом первого класса** в контексте конкретного ЯП, когда он:

- может быть связан с идентификатором;
- может быть передан в качестве аргумента процедурам и функциям;

Объект первого класса

Определение

Объект называют **объектом первого класса** в контексте конкретного ЯП, когда он:

- может быть связан с идентификатором;
- может быть передан в качестве аргумента процедурам и функциям;
- может быть возвращён в качестве результата;

Объект первого класса

Определение

Объект называют **объектом первого класса** в контексте конкретного ЯП, когда он:

- может быть связан с идентификатором;
- может быть передан в качестве аргумента процедурам и функциям;
- может быть возвращён в качестве результата;
- может быть создан во время исполнения программы;

Объект первого класса

Определение

Объект называют **объектом первого класса** в контексте конкретного ЯП, когда он:

- может быть связан с идентификатором;
- может быть передан в качестве аргумента процедурам и функциям;
- может быть возвращён в качестве результата;
- может быть создан во время исполнения программы;
- независим от именования (самоидентифициуем).

Объект первого класса

Определение

Объект называют **объектом первого класса** в контексте конкретного ЯП, когда он:

- может быть связан с идентификатором;
- может быть передан в качестве аргумента процедурам и функциям;
- может быть возвращён в качестве результата;
- может быть создан во время исполнения программы;
- независим от именования (самоидентифициуем).

Объект первого класса

Определение

Объект называют **объектом первого класса** в контексте конкретного ЯП, когда он:

- может быть связан с идентификатором;
- может быть передан в качестве аргумента процедурам и функциям;
- может быть возвращён в качестве результата;
- может быть создан во время исполнения программы;
- независим от именования (самоидентифициуем).

Примеры для C++ или Pascal:

- константы,

Объект первого класса

Определение

Объект называют **объектом первого класса** в контексте конкретного ЯП, когда он:

- может быть связан с идентификатором;
- может быть передан в качестве аргумента процедурам и функциям;
- может быть возвращён в качестве результата;
- может быть создан во время исполнения программы;
- независим от именования (самоидентифициуем).

Примеры для C++ или Pascal:

- константы,
- строки,

Объект первого класса

Определение

Объект называют **объектом первого класса** в контексте конкретного ЯП, когда он:

- может быть связан с идентификатором;
- может быть передан в качестве аргумента процедурам и функциям;
- может быть возвращён в качестве результата;
- может быть создан во время исполнения программы;
- независим от именования (самоидентифициуем).

Примеры для C++ или Pascal:

- константы,
- строки,
- структуры,

Объект первого класса

Определение

Объект называют **объектом первого класса** в контексте конкретного ЯП, когда он:

- может быть связан с идентификатором;
- может быть передан в качестве аргумента процедурам и функциям;
- может быть возвращён в качестве результата;
- может быть создан во время исполнения программы;
- независим от именования (самоидентифициуем).

Примеры для C++ или Pascal:

- константы,
- строки,
- структуры,
- объекты и т.д.

Связывание с идентификатором

Определив функцию в Pascal

```
function sqr(x: integer): integer;
begin
  sqr := x * x
end;
```

мы не сможем написать

```
f := sqr
```

Связывание с идентификатором

Определив функцию в Pascal

```
function sqr(x: integer): integer;
begin
  sqr := x * x
end;
```

мы не сможем написать

```
f := sqr
```

В языке Scheme это возможно:

```
> (define (sqr x) (* x x))

> (define f sqr)
> (f 4)
16
```

Связывание с идентификатором

Определив функцию в Pascal

```
function sqr(x: integer): integer;
begin
  sqr := x * x
end;
```

мы не сможем написать

```
f := sqr
```

В языке Scheme это возможно:

```
> (define (sqr x) (* x x))

> (define f sqr)
> (f 4)
16

> (define g *)
> (g 4 3)
12
```

Функции, как аргументы и результаты

Определим функцию выполняющую тождественное отображение:

```
> (define (id x) x)
```

Функции, как аргументы и результаты

Определим функцию выполняющую тождественное отображение:

```
> (define (id x) x)

> (id 4)
4
```

Функции, как аргументы и результаты

Определим функцию выполняющую тождественное отображение:

```
> (define (id x) x)

> (id 4)
4
```

Передадим ей в качестве аргумента функцию +:

```
> (id +)
<procedure:+>
```

Функции, как аргументы и результаты

Определим функцию выполняющую тождественное отображение:

```
> (define (id x) x)
```

```
> (id 4)
4
```

Передадим ей в качестве аргумента функцию +:

```
> (id +)
<procedure:+>
```

В качестве результата, мы снова получаем функцию:

```
> ((id +) 4 6)
10
```

Функции, как аргументы и результаты

Передавать функции-аргументы можно и в C++

C++:

```
typedef int TFun(int);

int dup(TFun f, int x)
{
  return f(f(x));
}

int sqr(int x)
{
  return x * x;
}

int z = dup(sqr, 2);
```

Чистота функций

Чем определяется значение функции?

Чистота функций

Чем определяется значение функции?

- В математике:
    - только фактическими аргументами.

Чистота функций

Чем определяется значение функции?

- В математике:
  - только фактическими аргументами.
- В программировании:
  - фактическими аргументами,
  - состоянием информационной среды.

Чистота функций

Чем определяется значение функции?

- В математике:
  - только фактическими аргументами.
- В программировании:
  - фактическими аргументами,
  - состоянием информационной среды.

Определение

Функция, не зависящая от состояния информационной среды и не меняющая её, называется **чистой функцией**.

Чистота функций

Чем определяется значение функции?

- В математике:
  - только фактическими аргументами.
- В программировании:
  - фактическими аргументами,
  - состоянием информационной среды.

Определение

Функция, не зависящая от состояния информационной среды и не меняющая её, называется **чистой функцией**.

Альтернативой чистой функции является **функция с побочным эффектом** или **разрушающая функция**.

Пример разрушающей функции

Pascal:

```
Var
  flag : Boolean;

Function f(n : Integer) : Integer;
begin
  If flag
    Then f := n;
    Else f := 2*n;
  flag := not(flag);
end;

Begin
  flag := True;
  Writeln( f(1) + f(2) );
  Writeln( f(2) + f(1) );
End.
```

Пример разрушающей функции

Pascal:

```
Var
  flag : Boolean;

Function f(n : Integer) : Integer;
begin
  If flag
    Then f := n;
    Else f := 2*n;
  flag := not(flag);
end;

Begin
  flag := True;
  Writeln( f(1) + f(2) );
  Writeln( f(2) + f(1) );
End.
```

Вывод программы:

```
5
4
```

Пример разрушающей функции

Pascal:

```
Var
  flag : Boolean;

Function f(n : Integer) : Integer;
begin
  If flag
    Then f := n;
    Else f := 2*n;
  flag := not(flag);
end;

Begin
  flag := True;
  Writeln( f(1) + f(2) );
  Writeln( f(2) + f(1) );
End.
```

Вывод программы:

```
5
4
```

С точки зрения математики, здесь две ошибки:

- $f(x) + f(y) \neq f(y) + f(x)$, хотя f целочисленная,

Пример разрушающей функции

Pascal:

```
Var
  flag : Boolean;

Function f(n : Integer) : Integer;
begin
  If flag
    Then f := n;
    Else f := 2*n;
  flag := not(flag);
end;

Begin
  flag := True;
  Writeln( f(1) + f(2) );
  Writeln( f(2) + f(1) );
End.
```

Вывод программы:

```
5
4
```

С точки зрения математики, здесь две ошибки:

- $f(x) + f(y) \neq f(y) + f(x)$, хотя f целочисленная,
- $flag = \mathrm{not}(flag)$ — абсурдное утверждение!

Чистота функции

Чистые функции

Функции с побочным эффектом

Чистота функции

Чистые функции

- для одинаковых аргументов, всегда дают одинаковые результаты;

Функции с побочным эффектом

Чистота функции

Чистые функции

- для одинаковых аргументов, всегда дают одинаковые результаты;

Функции с побочным эффектом

- могут возвращать различные результаты, в зависимости от контекста вызова;

Чистота функции

Чистые функции

- для одинаковых аргументов, всегда дают одинаковые результаты;
- их вычисление не зависит ни от истории вычислений, ни от его порядка;

Функции с побочным эффектом

- могут возвращать различные результаты, в зависимости от контекста вызова;

Чистота функции

Чистые функции

- для одинаковых аргументов, всегда дают одинаковые результаты;
- их вычисление не зависит ни от истории вычислений, ни от его порядка;

Функции с побочным эффектом

- могут возвращать различные результаты, в зависимости от контекста вызова;
- их вычисление зависит от порядка вычислений;

Чистота функции

Чистые функции

- для одинаковых аргументов, всегда дают одинаковые результаты;
- их вычисление не зависит ни от истории вычислений, ни от его порядка;
- для заданных аргументов, могут быть заменены результатом без вычисления;

Функции с побочным эффектом

- могут возвращать различные результаты, в зависимости от контекста вызова;
- их вычисление зависит от порядка вычислений;

Чистота функции

Чистые функции

- для одинаковых аргументов, всегда дают одинаковые результаты;
- их вычисление не зависит ни от истории вычислений, ни от его порядка;
- для заданных аргументов, могут быть заменены результатом без вычисления;
- могут быть полностью исключены из процесса вычисления, если результат не требуется.

Функции с побочным эффектом

- могут возвращать различные результаты, в зависимости от контекста вызова;
- их вычисление зависит от порядка вычислений;

Чистота функции

Чистые функции

- для одинаковых аргументов, всегда дают одинаковые результаты;
- их вычисление не зависит ни от истории вычислений, ни от его порядка;
- для заданных аргументов, могут быть заменены результатом без вычисления;
- могут быть полностью исключены из процесса вычисления, если результат не требуется.

Функции с побочным эффектом

- могут возвращать различные результаты, в зависимости от контекста вызова;
- их вычисление зависит от порядка вычислений;
- замена функции результатом или её исключение невозможны, если функция должна изменять состояние информационной среды.

Преимущества использования чистых функций

Верификация программ:

- возможность тестирования любых частей программы (функций) по отдельности;

Преимущества использования чистых функций

Верификация программ:

- возможность тестирования любых частей программы (функций) по отдельности;
- возможность заменять функции значениями и однозначно редуцировать программы;

Преимущества использования чистых функций

Верификация программ:

- возможность тестирования любых частей программы (функций) по отдельности;
- возможность заменять функции значениями и однозначно редуцировать программы;
- независимость результата работы функций от контекста и времени вызова позволяет полагаться на юнит-тестирование.

Преимущества использования чистых функций

Верификация программ:

- возможность тестирования любых частей программы (функций) по отдельности;
- возможность заменять функции значениями и однозначно редуцировать программы;
- независимость результата работы функций от контекста и времени вызова позволяет полагаться на юнит-тестирование.

Преимущества использования чистых функций

Верификация программ:

- возможность тестирования любых частей программы (функций) по отдельности;
- возможность заменять функции значениями и однозначно редуцировать программы;
- независимость результата работы функций от контекста и времени вызова позволяет полагаться на юнит-тестирование.

Оптимизация программ:

- однажды вычисленную для заданных аргументов функцию можно заменить её результатом;

Преимущества использования чистых функций

Верификация программ:

- возможность тестирования любых частей программы (функций) по отдельности;
- возможность заменять функции значениями и однозначно редуцировать программы;
- независимость результата работы функций от контекста и времени вызова позволяет полагаться на юнит-тестирование.

Оптимизация программ:

- однажды вычисленную для заданных аргументов функцию можно заменить её результатом;
- можно формально исключать неиспользуемые функции;

Преимущества использования чистых функций

Верификация программ:

- возможность тестирования любых частей программы (функций) по отдельности;
- возможность заменять функции значениями и однозначно редуцировать программы;
- независимость результата работы функций от контекста и времени вызова позволяет полагаться на юнит-тестирование.

Оптимизация программ:

- однажды вычисленную для заданных аргументов функцию можно заменить её результатом;
- можно формально исключать неиспользуемые функции;
- независимость результата функции от порядка вычисления и контекста позволяет легко производить параллельные вычисления.

Закрепление материала

- Являются ли объектами первого класса в языке Pascal

Закрепление материала

- Являются ли объектами первого класса в языке Pascal
  - символ (`Char`),

Закрепление материала

- Являются ли объектами первого класса в языке Pascal
  - символ (`Char`),
  - указатель,

Закрепление материала

- Являются ли объектами первого класса в языке Pascal
  - символ (`Char`),
  - указатель,
  - массив,

Закрепление материала

- Являются ли объектами первого класса в языке Pascal
  - символ (`Char`),
  - указатель,
  - массив,
  - тип?

Закрепление материала

- Являются ли объектами первого класса в языке Pascal
  - символ (`Char`),
  - указатель,
  - массив,
  - тип?
- Можно ли писать чистые функции и программы на языках C++ или Pascal?

1. Функция в математике
   - Определение
   - Способы описания
   - Некоторые специальные функции

2. Функции в программировании
   - Синтаксис применения и определения функций
   - Функции, как объекты первого класса
   - Чистота функций

3. Базовые принципы ФП
   - Переменные и присваивание
   - Программа, данные, процесс, результат

Присваивание и связывание

Присваивание и связывание

Требование использовать только чистые функции приводит к отказу от переменных и понятия присваивания.

Присваивание и связывание

Присваивание и связывание

Требование использовать только чистые функции приводит к отказу от переменных и понятия присваивания.

Присваивание заменяется на однократное **связывание** значения с символом.

Присваивание и связывание

Присваивание и связывание

Требование использовать только чистые функции приводит к отказу от переменных и понятия присваивания.

Присваивание заменяется на однократное **связывание** значения с символом.

Присваивание

```
i := 1;
```

Связывание

Присваивание и связывание

Присваивание и связывание

Требование использовать только чистые функции приводит к отказу от переменных и понятия присваивания.

Присваивание заменяется на однократное **связывание** значения с символом.

Присваивание

```
i := 1;
```

Связывание

```
(define i 1)
```

Присваивание и связывание

Присваивание и связывание

Требование использовать только чистые функции приводит к отказу от переменных и понятия присваивания.

Присваивание заменяется на однократное **связывание** значения с символом.

Присваивание

```
i := 1;

j := 4 + i;
```

Связывание

```
(define i 1)
```

Присваивание и связывание

Присваивание и связывание

Требование использовать только чистые функции приводит к отказу от переменных и понятия присваивания.

Присваивание заменяется на однократное **связывание** значения с символом.

Присваивание

```
i := 1;

j := 4 + i;
```

Связывание

```
(define i 1)

(define j (+ 4 i))
```

Присваивание и связывание

Присваивание и связывание

Требование использовать только чистые функции приводит к отказу от переменных и понятия присваивания.

Присваивание заменяется на однократное **связывание** значения с символом.

Присваивание

```
i := 1;

j := 4 + i;

k := k + 1;
```

Связывание

```
(define i 1)

(define j (+ 4 i))
```

Присваивание и связывание

Присваивание и связывание

Требование использовать только чистые функции приводит к отказу от переменных и понятия присваивания.

Присваивание заменяется на однократное **связывание** значения с символом.

Присваивание

```
i := 1;

j := 4 + i;

k := k + 1;
```

Связывание

```
(define i 1)

(define j (+ 4 i))

(define k (+ k 1))
```

Присваивание и связывание

Присваивание и связывание

Требование использовать только чистые функции приводит к отказу от переменных и понятия присваивания.

Присваивание заменяется на однократное **связывание** значения с символом.

Присваивание

```
i := 1;

j := 4 + i;

k := k + 1;
```

Связывание

```
(define i 1)

(define j (+ 4 i))

(define k (+ k 1))
```

Последнее определение вызовет сообщение об ошибке!

Переменные и присваивание

Что мы имеем в виду, когда пишем следующие выражения?

императивное программирование	математика и функциональное программирование

Переменные и присваивание

Что мы имеем в виду, когда пишем следующие выражения?

императивное программирование

математика и функциональное программирование

$$i = 5$$

Переменные и присваивание

Что мы имеем в виду, когда пишем следующие выражения?

императивное программирование

математика и функциональное программирование

i – **имя ячейки памяти**, в которую в данный момент записывается значение 5

$$i = 5$$

Переменные и присваивание

Что мы имеем в виду, когда пишем следующие выражения?

императивное программирование

математика и функциональное программирование

i – **имя ячейки памяти**, в которую в данный момент записывается значение 5

$$i = 5$$

i равно 5
(i – символ, обозначающий 5).

Переменные и присваивание

Что мы имеем в виду, когда пишем следующие выражения?

императивное программирование

математика и функциональное программирование

i – **имя ячейки памяти**, в которую в данный момент записывается значение 5

$$i = 5$$

i равно 5
(i – символ, обозначающий 5).

$$j = i$$

Переменные и присваивание

Что мы имеем в виду, когда пишем следующие выражения?

императивное программирование		математика и функциональное программирование
i – **имя ячейки памяти**, в которую в данный момент записывается значение 5	$i = 5$	i равно 5 (i – символ, обозначающий 5).
j – имя ячейки памяти, в которой записано значение из ячейки i **в момент присвоения.**	$j = i$	

Переменные и присваивание

Что мы имеем в виду, когда пишем следующие выражения?

императивное программирование		математика и функциональное программирование
i – **имя ячейки памяти**, в которую в данный момент записывается значение 5	$i = 5$	i равно 5 (i – символ, обозначающий 5).
j – имя ячейки памяти, в которой записано значение из ячейки i **в момент присвоения.**	$j = i$	j и i **всегда** равны друг другу (это одна и та же сущность).

Переменные и присваивание

Что мы имеем в виду, когда пишем следующие выражения?

императивное программирование		математика и функциональное программирование
i – **имя ячейки памяти**, в которую в данный момент записывается значение 5	$i = 5$	i равно 5 (i – символ, обозначающий 5).
j – имя ячейки памяти, в которой записано значение из ячейки i **в момент присвоения.**	$j = i$	j и i **всегда** равны друг другу (это одна и та же сущность).

$$i = i + 1$$

Переменные и присваивание

Что мы имеем в виду, когда пишем следующие выражения?

императивное программирование		математика и функциональное программирование
i – **имя ячейки памяти**, в которую в данный момент записывается значение 5	$i = 5$	i равно 5 (i – символ, обозначающий 5).
j – имя ячейки памяти, в которой записано значение из ячейки i **в момент присвоения.**	$j = i$	j и i **всегда** равны друг другу (это одна и та же сущность).
увеличение на единицу значения, **записанного прежде** в ячеку i.	$i = i + 1$	

Переменные и присваивание

Что мы имеем в виду, когда пишем следующие выражения?

императивное программирование		математика и функциональное программирование
i – **имя ячейки памяти**, в которую в данный момент записывается значение 5	$i = 5$	i равно 5 (i – символ, обозначающий 5).
j – имя ячейки памяти, в которой записано значение из ячейки i **в момент присвоения.**	$j = i$	j и i **всегда** равны друг другу (это одна и та же сущность).
увеличение на единицу значения, **записанного прежде** в ячеку i.	$i = i + 1$	Значение i неопределено или $i = \infty$ (конечного решения этого уравнения не существует).

Переменные и символы

Императивное программирование

Идентификаторы (переменные) — символы, ссылающиеся на ячейки памяти.

Переменные и символы

Императивное программирование

Идентификаторы (переменные) — символы, ссылающиеся на ячейки памяти.

В ячейках памяти могут быть те или иные объекты: данные (числа, строки, структуры и т.д.), процедуры (подпрограммы, функции и т.д.).

Переменные и символы

Императивное программирование

Идентификаторы (переменные) — символы, ссылающиеся на ячейки памяти.

В ячейках памяти могут быть те или иные объекты: данные (числа, строки, структуры и т.д.), процедуры (подпрограммы, функции и т.д.).

Объект жестко привязан к месту в памяти и к идентификатору. Потеря указателя на объект равносильна исчезновению объекта в контексте программы.

Переменные и символы

Императивное программирование

Идентификаторы (переменные) — символы, ссылающиеся на ячейки памяти.

В ячейках памяти могут быть те или иные объекты: данные (числа, строки, структуры и т.д.), процедуры (подпрограммы, функции и т.д.).

Объект жестко привязан к месту в памяти и к идентификатору. Потеря указателя на объект равносильна исчезновению объекта в контексте программы.

Математика и функциональное программирование

Идентификаторы — символы ссылающиеся на некие сущности (числа, объекты, функции и т.д.).

Переменные и символы

Императивное программирование

Идентификаторы (переменные) — символы, ссылающиеся на ячейки памяти.

В ячейках памяти могут быть те или иные объекты: данные (числа, строки, структуры и т.д.), процедуры (подпрограммы, функции и т.д.).

Объект жестко привязан к месту в памяти и к идентификатору. Потеря указателя на объект равносильна исчезновению объекта в контексте программы.

Математика и функциональное программирование

Идентификаторы — символы ссылающиеся на некие сущности (числа, объекты, функции и т.д.).

Все ссылки на некоторую величину эквивалентны самой этой величине.

Переменные и символы

Императивное программирование

Идентификаторы (переменные) — символы, ссылающиеся на ячейки памяти.

В ячейках памяти могут быть те или иные объекты: данные (числа, строки, структуры и т.д.), процедуры (подпрограммы, функции и т.д.).

Объект жестко привязан к месту в памяти и к идентификатору. Потеря указателя на объект равносильна исчезновению объекта в контексте программы.

Математика и функциональное программирование

Идентификаторы — символы ссылающиеся на некие сущности (числа, объекты, функции и т.д.).

Все ссылки на некоторую величину эквивалентны самой этой величине.

Тот факт, что на объект можно ссылаться под разными именами, никак не влияет на сам объект.

Переменные и символы

Императивное программирование

Идентификаторы (переменные) — символы, ссылающиеся на ячейки памяти.

В ячейках памяти могут быть те или иные объекты: данные (числа, строки, структуры и т.д.), процедуры (подпрограммы, функции и т.д.).

Объект жестко привязан к месту в памяти и к идентификатору. Потеря указателя на объект равносильна исчезновению объекта в контексте программы.

Математика и функциональное программирование

Идентификаторы — символы ссылающиеся на некие сущности (числа, объекты, функции и т.д.).

Все ссылки на некоторую величину эквивалентны самой этой величине.

Тот факт, что на объект можно ссылаться под разными именами, никак не влияет на сам объект.

Это свойство языков программирование называют **функциональностью** или **прозрачностью по ссылкам**.

Базовые принципы ФП

Таким образом, в функциональном программировании

- нет переменных,

Базовые принципы ФП

Таким образом, в функциональном программировании

- нет переменных,
- используются константы и символы;

Базовые принципы ФП

Таким образом, в функциональном программировании

- нет переменных,
- нет присваивания,
- используются константы и символы;

Базовые принципы ФП

Таким образом, в функциональном программировании

- нет переменных,
- нет присваивания,
- используются константы и символы;
- используется связывание;

Базовые принципы ФП

Таким образом, в функциональном программировании

- нет переменных,
- нет присваивания,
- нет заданного порядка вычисления,

- используются константы и символы;
- используется связывание;

Базовые принципы ФП

Таким образом, в функциональном программировании

- нет переменных,
- нет присваивания,
- нет заданного порядка вычисления,

- используются константы и символы;
- используется связывание;
- вычисления описываются, как определения функций.

Базовые принципы ФП

Таким образом, в функциональном программировании

- нет переменных,
- нет присваивания,
- нет заданного порядка вычисления,
- используются константы и символы;
- используется связывание;
- вычисления описываются, как определения функций.

Основной принцип ФП

Функциональная парадигма предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, без явного хранения состояния программы.

Базовые принципы ФП

Программа

Функциональная программа представляет собой композицию чистых функций.

Базовые принципы ФП

Программа

Функциональная программа представляет собой композицию чистых функций.

Процесс

Вычисление функционального выражения состоит в подстановке фактических аргументов вместо формальных.

Базовые принципы ФП

Программа

Функциональная программа представляет собой композицию чистых функций.

Процесс

Вычисление функционального выражения состоит в подстановке фактических аргументов вместо формальных.

Данные

Исходные данные — фактические аргументы функций.

Базовые принципы ФП

Программа

Функциональная программа представляет собой композицию чистых функций.

Процесс

Вычисление функционального выражения состоит в подстановке фактических аргументов вместо формальных.

Данные

Исходные данные — фактические аргументы функций.

Результат

Результат работы программы однозначно и исключительно определяется входными данными.

Закрепление материала

- Как должна выглядеть функциональная (прозрачная по ссылкам) программа на C++?

Закрепление материала

- Как должна выглядеть функциональная (прозрачная по ссылкам) программа на C++?

```
Type f(...)
{
  const ...
  ...
  return ...
}

void main(void)
{
  <input> arg1, arg2, ...
  <output> f(..., g(...), ...);
}
```

Закрепление материала

- Как должна выглядеть функциональная (прозрачная по ссылкам) программа на C++?

```
Type f(...)
{
  const ...
  ...
  return ...
}

void main(void)
{
  < input > arg1, arg2, ...
  < output > f(..., g(...), ...);
}
```

- Чему будет равно значение `i`, если команда

```
int i = i + 1
```

будет первой в программе на C++?

Закрепление материала

- Как должна выглядеть функциональная (прозрачная по ссылкам) программа на C++?

```
Type f(...)
{
  const ...
  ...
  return ...
}

void main(void)
{
  < input > arg1, arg2, ...
  < output > f(..., g(...), ...);
}
```

- Чему будет равно значение `i`, если команда

```
int i = i + 1
```

будет первой в программе на C++?

- Чем, с точки зрения функциональности, оправдан префиксный синтаксис языка Scheme?

Закрепление материала

- Как должна выглядеть функциональная (прозрачная по ссылкам) программа на C++?

```
Type f(...)
{
  const ...
  ...
  return ...
}

void main(void)
{
  <input> arg₁, arg₂, ...
  <output> f(..., g(...), ...);
}
```

- Чему будет равно значение `i`, если команда

  `int i = i + 1`

  будет первой в программе на C++?
- Чем, с точки зрения функциональности, оправдан префиксный синтаксис языка Scheme?
- Можно ли построить процессор, основываясь на функциональной парадигме?