Розділ 1

**Синтаксис у двох словах**

Синтаксис Pharo дуже близький до синтаксису його предка, Smalltalk. Синтаксис такий, що програмний код при читанні схожий на спрощену англійську.

Наступний метод класу Week демонструє приклад синтаксису. Він перевіряє чи DayNames містить аргумент, тобто чи аргумент є правильною назвою дня тижня. Якщо так, то цей аргумент присвоюється змінній StartDay.

startDay: aSymbol

(DayNames includes: aSymbol)

ifTrue: [ StartDay := aSymbol ]

ifFalse: [ self error: aSymbol, ' is not a recognised day name' ]

Синтаксис у Pharo простий. По суті, складається з відправки повідомлення (вирази). Складаються вирази з невеликого числа примітивних елементів. Є 6 ключових слів, і немає синтаксису для керування структурами та створення нових класів. Замість цього використовується відправка повідомлень об'єктам. Наприклад, замість керуючої структури if-then-else використовується посилка повідомлення ifTrue: логічному (Boolean) об'єкту. Нові класи створюються відправленням повідомлення суперкласу.

* 1. **Синтаксичні елементи**

Вирази будуються за допомогою наступних елементів:

1. Шість зарезервованих ключових слів або *псевдо змінних*: self, super, nil, true, false, і thisContext
2. Константні вирази літеральних об’єктів, що включають числа, літери, рядки, символи та масиви
3. Оголошення змінної
4. Присвоювання
5. Блоки повідомлень
6. Повідомлення

Нижче наведено таблицю з прикладами різних синтаксичних елементів.

|  |  |
| --- | --- |
| **Синтаксис** | **Що означає** |
| startPoint | Назва змінної |
| Transcript | Назва глобальної змінної |
| self | Псевдо змінна |
| 1 | Десяткове ціле число |
| 2r101 | Двійкове число |
| 1.5 | Дійсне число |
| 2.4e7 | Експоненціальний запис числа |
| $a | Літера ‘а’ |
| ‘Hello’ | Рядок ‘Hello’ |
| #Hello | Символ #Hello |
| #(1 2 3) | Літеральний масив |
| { 1. 2. 1+2} | Динамічний масив |
| “a comment” | Коментар |
| |xy| | Тимчасові змінні |
| X:=1 | Присвоювання 1 змінній х |
| [:x|x+2] | Блок, який обчислює х+2 |
| <primitive: 1> | Віртуальний примітивний метод |
| 3 factorial | Унарне повідомлення |
| 3 + 4 | Бінарне повідомлення |
| 2 raisedTo: 6 module 10 | Ключове повідомлення |
| ^ true | Повернення значення true |
| Transcript show: ‘hello’. Transcript cr | Розділювач виразів (.) |
| Transcript show: ‘hello’;cr | Розділювач каскадних повідомлень (;) |

**Локальні змінні.** startPoint — це ім’я змінної або ідентифікатор. Умовно, ідентифікатори складаються зі слів в “camelCase” (тобто кожне слово, за виключенням першого, починається з великої літери). Перша літера змінної екземпляра, метода, аргумента блоку чи тимчасової змінної повинна бути малою. Це вказує читачеві, що змінна в приватній області видимості.

**Загальні змінні**. Ідентифікатори загальних змінних починаються з великої літери і до них відносяться глобальні змінні, змінні класу, словники та назви класів. Transcript є глобальною змінною і екземпляром класу TranscriptStream.

**Отримувач**. Ключове слово self посилається на об’єкт поточного метода, який виконується. Ми називаємо його “Отримувачем” тому, що це об’єкт, на який посилається self, є отримувачем повідомлення. self називається псевдо змінною так як ми не можемо призначити цій змінній інший об’єкт.

**Цілочислові**. В доповнення до звичайних десяткових цілих чисел, наприклад 42, Pharo використовує і інші системи числення. 2r101 - це 101 у двійковій системі числення (бінарній), яке рівне десятковому числу 5.

**Дійсні числа**. Можуть записуватися з використанням степені числа 10: 2.4e7 — це 2.4 × 10 7.

**Літерали**. Знак долара і наступний літерал позначають екземпляр класу Character, наприклад $a. Недруковані символи можна отримати, відправивши відповідні повідомлення класу Character, такі як Character space або Character tab.

**Рядки**. Одинарні лапки використовуються для позначення екземпляра класу String, який складається із послідовності букв и символів. Якщо потрібно в рядок (String) додати одинарну лапку, то її потрібно продублювати, наприклад 'G ''day '.

**Символи**. так як і рядки (Strings), містять послідовності букв. Але вони відрізняються від рядків тим, що вони є унікальними. В системі може бути тільки один символьний об’єкт #hello, а рядків 'hello ' скільки завгодно.

**Масиви, які створюються на етапі компілювання**. Визначаються за допомогою #() і між дужками вводять літеральні об’єкти, розділюючи їх пробілами. Все, що є між дужками, створюється під час компіляції. Наприклад, #(27 (true false) abc) складається з трьох елементів: цілого числа 27, масиву, який містить два логічних об’єкти и символу #abc. Цей самий приклад можна записати по-іншому: #(27 #(true false) #abc).

**Масиви, які створюються на етапі виконання**. Фігурні дужки {} визначають (динамічний) масив на етапі виконання. Елементи масиву – вирази, що розділяються крапкою. Наприклад, {1. 2. 1+2.} визначає масив з трьох елементів: 1, 2 и результат виразу 1+2.

**Коментарі**. Поміщені у подвійні лапки “ ”. "hello" – це коментар, не рядок (String) і ігнорується компілятором Pharo. Коментарі можуть містити кілька рядків.

**Визначення локальних змінних**. Вертикальні полоски | | містять оголошення однієї чи кількох локальних змінних в методі (також і в блоці).

**Присвоювання обєкта змінній** здійснюється оператором :=.

**Блоки**. Квадратні дужки [] визначають блок (block), також відомий як block closure чи lexical closure, обєкт який представляє собою функцію. Як ми побачимо далі, блоки можуть мати аргументи і локальні змінні.

**Примітиви**. <primitive: ...> позначає посилання на примітив віртуальної машини. Наприклад, <primitive: 1> - це VM примітив для Smallinteger. Весь код, який розташований за примітивом, виконується тільки, якщо примітив закінчився з помилкою. Аналогічний синтакс використовується для анотації методів.

**Унарні повідомлення**. Вони складаються з одного слова (наприклад, factorial), яке посилається отримувачу (наприклад 3). У виразі «3 factorial» 3 це отримувач, а factorial це селектор повідомлення.

**Бінарні повідомлення**. Це оператори (+), які посилаються отримувачу і приймають один аргумент. У виразі 3 + 4 отримувачем є 3, селектором повідомлення є + і аргументом є 4. Результатом такого виразу є ціле число 7.

**Ключові повідомлення**. Складаються з кількох слів (наприклад, raisedTo:modulo:), кожне з яких закінчується двокрапкою і приймає один аргумент. У виразі «2 raisedTo: 6 modulo: 10» селектор повідомлення «raisedTo:modulo:» приймає два аргументи 6 и 10, які розміщені після двокрапки. Повідомлення відправляється отримувачу 2.

**Повернення значення з метода**. Оператор ^ використовується для повернення відповіді з метода.

**Послідовності повідомлень**. Крапка (.) є розділювачем повідомлень. Якщо між двома виразами поставити крапку, то вони перетворюються у два незалежні повідомлення.

**Каскадні повідомлення**. Крапка з комою (;) використовується для відправки кількох повідомлень одному отримувачу. У виразі Transcript show: 'hello '; cr. спочатку відправляється повідомлення show: 'hello ' отримувачу Transcript, після цього відправляється унарне повідомлення cr цьому ж отримувачу.

Класи Number, Character, String I Boolean описані більш детально у розділі 10: Базові класи.

* 1. **Псевдо змінні.**

У Pharo є 6 зарезервованих слів або псевдо змінних: nil, true, false, self, super та thisContext. Вони називаються псевдо змінними тому, що їх значення наперед визначені і не можуть бути змінені. true, false i nil є константами, а значення змінних self, super та thisContext є динамічними, бо залежать від коду.

* ***true*** i ***false*** є унікальнми екземплярами логічних класів True та False відповідно. Див. Розділ 10: Базові класи.
* ***self*** завжди посилається на отримувача поточно виконуваного метода.
* ***super*** також посилається на отримувача поточного метода, але відмінність в тому, що ми посилаємо повідомлення суперкласу. Для більш детальної інформації дивіться Розділ 6: Обєктна модель Pharo.
* ***nil*** це невизначений обєкт. Є унікальним екземпляром класу UndefinedObject. Змінні екземплярів, класів і локальні змінні при ініціалізації встановлюються в значення nil.
* ***thisContext*** це псевдо змінна, яка представляє верхню частину стеку виконання. thisContext переважно не представляє інтересу для більшості програмістів, але він потрібен для розробки інструментів, таких як відладжувач (debugger), та для реалізації виключень.
  1. **Відправка повідомлення.**

Є три типи повідомлень у Pharo:

1. Унарні повідомлення складаються тільки з селектора без аргументів. «1 factorial» - відправляє повідомлення factorial обєкту 1.
2. Бінарні повідомлення складаються з селектора та одного аргумента. «1+2» - відправляє повідомлення + з аргументом 2 обєкту 1.
3. Ключові повідомлення складаються з довільної кількості аргументів. «2 raisedTo: 6 modulo: 10» - відправляє повідомлення, яке складається з селектора raisedTo:modulo і аргументів 6 та 10, обєкту 2.

Селектори унарних повідомлень складаються з букв та цифр і починаються з малої букви.

Селектори бінарних повідомлень складаються з одного з наступних символів: +=/\\*~<>=@%|&?,

Селектори ключових повідомлень складаються з серії ключових слів, кожне з яких складається з букв, цифр і починається маленькою буквою і закінчується двокрапкою.

Унарні повідомлення мають вищий пріоритет ніж бінарні, а бінарні вищий ніж ключові повідомлення.

2 raisedTo: 1 + 3 factorial -> 128

Спочатку ми надсилаємо factorial обєкту 3, потім надсилаємо + 6 обєкту 1, і потім надсилаємо raisedTo: 7 обєкту 2.

Зауважимо, що обчислення виразу відбувається зліва направо, тому:

1 + 2 \* 3 -> 9.

Для зміни порядку обчислення використовують круглі дужки:

1 + (2 \* 3) –> 7

Надсиалння повідомлень може комбінуватися з крапкою та крапкою з комою. Послідовність виразів розділених крапкою буде обчислюватись по черзі.

Transcript cr.

Transcript show: ‘hello world’.

Transcript cr.

Спочатку відправиться повідомлення cr обєкту Transcript, потім відправиться show: ‘hello world’ і потім cr.

Коли повідомлення відправляється одному і тому ж отримувачу, то краще використовувати каскадні повідомлення. Отримувач описується один раз, а послідовність повідомлень розділяється крапкою з комою.

Transcript cr; show: ‘hello world’; cr

* 1. **Синтаксис методу.**

Слід зауважити, що вирази виконуються всюди (workspace, debugger, browser і т.д.) а методи зазвичай створюються (редагуються) в браузері класів, або у відладчику (debugger). Методи також можна створювати з файлів, але такий шлях використовується рідко.

Програми розробляються зміною або створенням нових методів у певному класі. (Новий підклас створюється відправленням повідомлення існуючому класу).

Наведений нижче метод lineCount розташований у класі String. (Далі при описі методів ми будемо використовувати запис виду ClassName >> MethodName, тому ми викликаємо цей метод так – String >> lineCount).

String >> lineCount

"Answer the number of lines represented by the receiver, where every cr adds

one line."

| cr count |

cr := Character cr.

count := 1 min: self size.

self do:

[:c | c == cr ifTrue: [count := count + 1]].

^ count

Синтаксично метод складається з:  
1. заголовка методу, що включає в себе ім'я (ie, lineCount) і аргументів (відсутні в даному прикладі);  
2. коментарів (можуть розташовуватися в будь-якій частині методу, але за згодою зазвичай йдуть після заголовка і пояснюють, що робить метод);  
3. визначення локальних змінних;  
4. кількох пропозицій розділеними крапками, в даному випадку 4.

На початку пропозиції може розташовуватися символ ^, який означає, що після виконання даної пропозиції, метод поверне відправнику результат. При відсутності в методі символу ^, одержувач отримає як результат значення self.

Аргументи і локальні змінні слід завжди починати з малої літери. Імена, що починаються з великої літери, використовуються для глобальних змінних. Імена класів, наприклад Character, є звичайними глобальними змінними і мають посилання на об'єкт, який представляє клас.

* 1. **Синтаксис блока.**

Блоки забезпечують механізм відкладеної послідовності дій. Блок, по суті, анонімна функція. Блок обчислюється, посилаючи повідомлення value. Результатом виконання зазвичай є значення останнього виразу в тілі блока, але тільки якщо немає явного повернення результату (^).

[1 + 2] value → 3

Блоки можуть мати аргументи, які починаються з символу (:) двокрапки. Вертикальна смуга відокремлює тіло блоку від аргументів. При виконанні блоку з одним аргументом, потрібно відправити повідомлення value: argument. Якщо аргументів у блоку 2, тоді повідомлення зміниться на value: argument1 value: argument2 і т.д. до 4 аргументів.

[: X | 1 + x] value: 2 → 3  
[: X: y | x + y] value: 1 value: 2 → 3

Якщо блок більш ніж з чотирма параметрами, тоді треба використовувати повідомлення valueWithArguments: argumentArray, де argumentArray - масив аргументів. (Не рекомендується використовувати блоки з великим числом параметрів).

Блок дозволяє визначати локальні змінні, які розділяються вертикальними смугами як і при декларуванні змінних методу. Розташовані вони після аргументів блоку.

[: x: y | | z | z: = x + y. z] value: 1 value: 2 → 3

Блоки можуть посилатися на змінні розташовані поза нього. У наступному прикладі блок посилається на змінну x розташовану в методі, не в блоці.

| x |  
x: = 1.  
[: y | x + y] value: 2 → 3

Також вони є екземплярами класу BlockClosure. Це означає, що вони є об'єктами і можуть бути приєднані до змінної і відправлені як аргумент будь-якого об'єкта.

* 1. **Порівняння і цикли у двох словах.**

У Фаро відсутній спеціальний синтаксис для керуючих структур. Замість них використовуються пропозиції з посилкою повідомлень булевим, числовим і колекціям. А в якості аргументів використовуються блоки.

Порівняння - це пропозиція, де надсилається повідомлення ifTrue :, ifFalse: або ifTrue: ifFalse результату булевого виразу. Більш докладно про операції порівняння у розділі 8.

(17 \* 13> 220)  
ifTrue: ['bigger']  
ifFalse: ['smaller'] → 'bigger'

Цикли теж реалізовані через відправку повідомлень блокам, цілочисельним або колекціям. Так як в циклах умови виходу можуть бути обраховані повторно, то вони сконструйовані за допомогою блоків а не булевих об'єктів. Нижче представлений приклад циклу whileTrue.

n: = 1.  
[N <1000] whileTrue: [n: = n \* 2].  
n → 1024

whileFalse - поки хибна умова триває цикл.

n: = 1.  
[N> 1000] whileFalse: [n: = n \* 2].  
n → 1024

timesRepeat: - повторити тіло циклу фіксовану кількість разів.

n: = 1.  
10 timesRepeat: [n: = n \* 2].  
n → 1024

Також можна надсилати повідомлення to: do: числу для створення циклу з параметром. Початкове значення циклу - одержувач, перший аргумент  
повідомлення - кінцеве значення циклу, другий - тіло циклу.

result: = String new.  
1 to: 10 do: [: n | result: = result, n printString, ''].  
result → '1 2 3 4 5 6 7 8 9 10'

**Ітератори вищих порядків**. Колекції складаються з великої кількості класів, багато з яких мають однаковий протокол. Більшість повідомлень для ітерацій складаються з: do, collect, select :, reject :, detect: і inject: into :. Ці повідомлення визначають ітерації вищих порядків, які дозволяють писати більш компактний код.  
**Interval** - колекція, яка дозволяє ітерувати послідовність чисел від початкового числа і до кінцевого. 1 to: 10 реалізує інтервал від 1 до 10. Так як Interval є  
колекцією то можна послати повідомлення do :. Як аргумент використовується блок, який виконує певні дії для кожного елемента колекції.

result: = String new.  
(1 to:10) do: [:n | result:= result, n printString, ''].  
result → '1 2 3 4 5 6 7 8 9 10'

**collect**: створює нову колекцію такого ж розміру, трансформуючи кожен елемент.

(1 to: 10) collect: [: each | each \* each] → # (1 4 9 16 25 36 49 64 81 100)

**select**: і **reject**: створює нову колекцію, що включає в себе підмножину елементів, які задовольняють (чи ні) булевий блоку порівняння.

**detect**: повертає перший елемент, який задовольняє умову. Слід зауважити, що рядки теж є колекціями (символів), тому можна ітерувати по всіх символах.

'Hello there' select: [: char | char isVowel] → 'eoee'  
'Hello there' reject: [: char | char isVowel] → 'hll thr'  
'Hello there' detect: [: char | char isVowel] → $e

* 1. **Примітиви і прагми.**

У Pharo все є об'єктом і вся взаємодія між об'єктами здійснюється посилкою повідомлень. Проте певні об'єкти можуть викликати віртуальні машинні примітиви.

Наприклад, всі нижче наведені повідомлення реалізовані як примітиви:   
- виділення пам'яті (new, new :)  
- маніпуляції бітами (bitAnd :,ditOr :, ditShift :)  
- точки (Point) та целочисленная арифметика (+ - <> \* /= ==. . . )  
- доступ до масивів (at :, at: put).

Синтаксис при виклику примітиву такий <primitive: aNumber>. Метод який викликає примітив також може містити Smalltalk код, який отримає управління, якщо примітив завершитися помилкою.

Нижче показаний код методу SmallInteger >> +. Якщо примітив завершиться помилкою, то виконається наступна пропозиція super + aNumber.  
  
+ ANumber  
"Primitive. Add the receiver to the argument and answer with the result  
if it is a SmallInteger. Fail if the argument or the result is not a  
SmallInteger Essential No Lookup. See Object documentation  
whatIsAPrimitive. "  
<primitive: 1>  
^ super + aNumber

У Pharo подібний синтаксис також використовується для анотацій методу - прагми.

* 1. **Короткий огляд розділу.**
* Pharo використовує тільки 6 зарезервованих ідентифікаторів, які називаються псевдозмінними: true, false, nil, self, super, thisContext.
* Є 5 видів літеральних обєктів: числа (5, 2.5, 1.9, е15, 2r111), літери ($a), рядки (‘hello’), символи (#abc) і масиви (#(‘hello’, #ab)).
* Рядки (String) поміщаються в одинарні лапки, коментарі у подвійні. Одинарні лапки всередині рядка дублюються.
* Подібні рядкам, сиволи унікальні у середовищі.
* Для визначення літерального масиву використовується #(…), а для динамічного масиву – {…}. Потрібно зауважити, що #(1 + 2) size -> 3, a {1 + 2} size -> 1.
* Повідомлення бувають трьох видів : унарні (1 asString, Array new), бінарні (1+2) та ключові (‘hi’ at: 1 put: #a).
* Каскадні повідомлення – послідовність повідомлень одному отримувачу, розділені крапкою з комою: OrderedCollection new add: #calvin; add: #hobbes; size -> 2.
* Локальні змінні поміщаються у вертикальні смуги. := використовується для присвоєння: |x| x:= 1.
* Вирази складаються з відправки повідомлень, каскадних повідомлень і присвоєнь, які обчислюються зліва на право. Також можна використовувати дужки для групування або зміни пріоритету.
* Блоки – вирази заключені у квадратні дужки. Можуть мати аргументи і тимчасові змінні. Вираз у блоці не виконується поки йому не буде відправлено повідомлення value з правильною кількість аргументів.  
  [:x | x + 2] value: 4 -> 6.
* Відсутній вбудований синтаксис для керуючих структур, все реалізовано через відправку повідомлень з використанням блоків.