Ruby 치트시트

기본 구문 및 변수

- 주석: `#`로 시작.
- 출력: `puts` (줄바꿈 포함), `print` (줄바꿈 미포함).
- 변수: 타입 선언 없이 이름에 값을 할당. `my_variable = "Hello"`
- 상수: 대문자로 시작. `MY_CONSTANT = 3.14`
- 심볼 (Symbol): `:my_symbol`. 문자열과 비슷하지만 불변이며 효율적.
- 문자열 보간: `"Hello, #{my_variable}"`

제어 흐름

- `if-elsif-else-end`:

```
if condition
  # ...
elsif another_condition
  # ...
else
  # ...
end
```

- 후행 `if`/`unless`: `puts "It's true" if true`
- `unless`: `if`의 반대. `unless condition ... end`
- `case-when-end`:

```
case variable
when "value1"
  # ...
when "value2"
  # ...
else
  # ...
end
```

- `while`/`until` 루프: `while condition ... end`
- `for` 루프: `for item in collection ... end` (잘 사용되지 않음)

메서드 (Methods)

- 메서드 정의:

```
def my_method(param1,
param2="default")
  # 메서드 본문
  param1 + param2 # 마지막 표현식이 암시적으로 반환됨
end
```

- `return`: 명시적으로 값을 반환할 때 사용.
- 메서드 이름의 `?`와 `!`: `?`는 불리언 값을 반환하는 관례, `!`는 객체를 직접 수정(destructive)하거나 위험한 작업을 수행하는 관례.

블록, Proc, 람다

Ruby의 강력한 기능으로, 코드 블록을 메서드에 전달할 수 있습니다.

- 블록 (Block): `do ... end` 또는 `{ ... }`.

```
[1, 2, 3].each do |number|
  puts number
end
```

- `yield`: 메서드 내에서 전달된 블록을 실행.
- Proc: 블록을 객체로 만든 것. `my_proc = Proc.new { |x| puts x }`
- 람다 (Lambda): Proc과 유사하지만, 인자 수에 엄격하고 `return` 동작이 다름. `my_lambda = ->(x) { puts x }`

클래스와 객체

모든 것은 객체입니다.

- 클래스 정의:

```
class MyClass
  # 생성자
  def initialize(name)
    @name = name # 인스턴스 변수
  end

  # 인스턴스 메서드
  def greet
    "Hello, #{@name}!"
  end
end
```

- 객체 생성: `obj = MyClass.new("Ruby")`
- 접근자 (Accessors):
  - `attr_reader :name` (getter)
  - `attr_writer :name` (setter)
  - `attr_accessor :name` (getter & setter)
- 상속: `class Derived < Base ... end`
- 모듈 (Modules): 네임스페이스를 만들거나 클래스에 믹스인(mixin)하여 다중 상속처럼 사용.

```
module MyModule
  def fly
    puts "I'm flying!"
  end
end

class Bird
  include MyModule
end
```

주요 컬렉션

Array

```
arr = [1, "apple", 3.5]
arr << "new_item" # 추가
first = arr[0] # 접근
arr.each { |item| puts item } # 반복
```

Hash

키-값 쌍. (다른 언어의 딕셔너리)

```
my_hash = { "name" => "Jules", "age" => 30 }
my_hash = { name: "Jules", age: 30 } # Ruby 1.9+ 심볼 키

name = my_hash[:name] # 접근
my_hash[:city] = "Seoul" # 추가/수정
```

Enumerable 모듈

`Array`, `Hash` 등 많은 컬렉션 클래스가 포함하는 강력한 반복 메서드 모음.

- `map` / `collect`: 각 요소에 블록을 적용한 새 배열 반환.
- `select` / `filter`: 블록이 `true`를 반환하는 요소만 포함하는 새 배열 반환.
- `reject`: `select`의 반대.
- `reduce` / `inject`: 누적 연산.
- `any?`, `all?`, `none?`, `one?`

RubyGems 및 Bundler

- RubyGems: Ruby의 패키지 관리자.
  - `gem install <gem_name>`: 젬 설치.
  - `gem list`: 설치된 젬 목록.
- Bundler: 프로젝트의 젬 종속성 관리.
  - `Gemfile`에 필요한 젬을 명시.
  - `bundle install`: `Gemfile`에 명시된 젬 설치.
  - `bundle exec <command>`: `Gemfile`의 컨텍스트에서 명령어 실행.

예외 처리

```
begin
  # 예외가 발생할 수 있는 코드
rescue SomeError => e
  # 특정 예외 처리
  puts "Error: #{e.message}"
rescue
  # 그 외 모든 예외 처리
ensure
  # 예외 발생 여부와 관계없이 항상 실행
end
```