Julia 치트시트

기본 문법

Julia는 고성능 수치 계산 및 데이터 과학을 위해 설계된 현대적인 프로그래밍 언어입니다. 파이썬의 편리함과 C의 성능을 동시에 지향합니다.

변수 정의

Julia는 동적 타이핑을 지원하지만, 필요한 경우 명시적으로 타입을 지정할 수도 있습니다.

```
x = 10              # 정수형 (Int64)
y = "안녕하세요!"    # 문자열 (String)
z = 3.14            # 실수형 (Float64)
```

함수 정의 및 호출

함수는 `function` 키워드를 사용하여 정의하며, 마지막으로 평가된 값이 자동으로 반환됩니다.

```
# 표준적인 함수 정의
function greet(name)
  return "안녕하세요, $name!"
end

# 짧은 형식의 함수 정의 (Assignment form)
add(x, y) = x + y

# 함수 호출
println(greet("Julia"))
```

제어 흐름

If-Else 조건문

```
a = 5
if a > 0
  println("양수입니다")
elseif a < 0
  println("음수입니다")
else
  println("0입니다")
end
```

반복문 (For & While)

```
# For 루프: 지정된 범위나 컬렉션을 순회합니다.
for i in 1:5
  println(i)
end

# While 루프: 조건이 참인 동안 반복 실행합니다.
i = 1
while i ⩽ 5
  println(i)
  i += 1
end
```

데이터 구조

배열 (Arrays)

Julia의 배열 인덱스는 **1부터 시작**하며, 다차원 배열 처리에 최적화되어 있습니다.

```
# 1차원 배열 (Vector)
arr = [1, 2, 3, 4, 5]

# 다차원 배열 (Matrix)
# 공백으로 열을 구분하고 세미콜론으로 행을 구분합니다.
matrix = [1 2; 3 4]
```

딕셔너리 (Dictionaries)

키(Key)와 값(Value)의 쌍을 저장하는 가변적인 컬렉션입니다.

```
# 딕셔너리 생성
dict = Dict("이름" ⇒ "Julia", "버전" ⇒ 1.9)

# 요소 접근
println(dict["이름"])
```

문자열 (Strings)

문자열 연결에는 `*` 연산자를 사용하며, `$`를 이용한 문자열 보간이 가능합니다.

```
str1 = "Hello"
str2 = "Julia"

# 문자열 연결 (Concatenation)
combined_str = str1 * " " * str2

# 문자열 보간 (Interpolation)
name = "Alice"
message = "안녕하세요, $(name)님!"
```

튜플 (Tuples)

한 번 생성하면 내용을 변경할 수 없는 고정된 크기의 컬렉션입니다.

```
# 튜플 생성 및 접근
tup = (1, "hello", 3.14)
println(tup[2]) # "hello"

# 튜플 언패킹 (Unpacking)
a, b, c = tup
```

구조체 (Structs)

사용자 정의 데이터 타입을 정의할 때 사용합니다. 기본적으로는 불변(immutable)입니다.

```
# 불변 구조체 정의
struct Point
  x::Int
  y::Int
end

# 가변 구조체가 필요한 경우 'mutable struct'를 사용합니다.
mutable struct MutablePoint
  x::Int
end

# 인스턴스 생성 및 속성 접근
p = Point(10, 20)
println(p.x)
```

고급 기능

에러 처리 (Exception Handling)

`try-catch` 블록을 사용하여 런타임 에러를 안전하게 관리합니다.

```
function divide(a, b)
  try
    return a / b
  catch e
    if isa(e, DivideError)
      println("0으로 나눌 수 없습니다!")
    else
      println("에러 발생: $e")
    end
  end
end
```

모듈 및 패키지 관리

`using` 키워드로 모듈을 불러오며, `Pkg`를 통해 패키지를 관리합니다.

```
# 표준 라이브러리 또는 설치된 모듈 사용
using LinearAlgebra
println(det([1 2; 3 4])) # 행렬식 계산

# 패키지 추가 (REPL에서는 ']' 키를 눌러 관리 모드 진입 가능)
# using Pkg
# Pkg.add("DataFrames")
```

브로드캐스팅 (Broadcasting)

점(`.`) 연산자를 사용하여 배열의 각 요소에 함수나 연산을 개별적으로 적용하는 강력한 기능입니다.

```
# 배열의 모든 요소에 10을 더함
arr = [1, 2, 3]
result = arr .+ 10
println(result) # [11, 12, 13]
```

컴프리헨션 (Comprehensions)

기존 컬렉션을 바탕으로 새로운 배열이나 딕셔너리를 간결하게 생성합니다.

```
# 배열 컴프리헨션
squares = [i^2 for i in 1:5] # [1, 4, 9, 16, 25]

# 딕셔너리 컴프리헨션
dict_comp = Dict(i ⇒ i^2 for i in 1:3)
```

유용한 팁

```
# 도움말 모드: REPL에서 '?'를 입력한 후 함수명을 치면 문서를 보여줍니다.
# ?println

# 패키지 상태 확인
using Pkg
Pkg.status()
```