AviationScript Language

Davi Reis Vieira de Souza

Motivação

Por que AviationScript?

- 1 - Linguagem voltada para definir rotas e instruções para aeronaves.
- 2 - Facilita a programação e a automação de tarefas no contexto da aviação.
- 3 – Adoro aviação desde minha infância, e gostaria de poder envolver este hobby com meu projeto de Lógica de Computação.
- 4 – Gera de forma mais rápida rotas de voo e ajuda em cálculos que precisam ser feitos durante a preparação de voo.

Características

Sobre a linguagem

- 1 - Linguagem de programação estruturada.
- 2 - Tipagem estática.
- 3 - Sintaxe simples e intuitiva.
- 4 - Possui comandos específicos para configuração de rotas e waypoints.
- 5 - Funções matemáticas para cálculos relacionados à aviação.

Características

Sobre a linguagem

- 6 – Linguagem tipada, possuindo operações com inteiros, floats e strings.
- 7 – Uso de condições na criação de códigos.
- 8 – Criação de loops com while.
- 9 – Uso de funções para organização de código.

Características

EBNF

```
BLOCK → { STATEMENT };

STATEMENT → ( λ | ASSIGNMENT | PRINT | WHILE | IF | FUNCTION | AVIATION_FUNC | MATH_FUNC | RETURN ), "\n";

ASSIGNMENT → IDENTIFIER, ( CREATING | SETTING, CALLFUNC );

CREATING → "::", TYPE, [ "=", RELEXPR ];

TYPE → "Int" | "Float" | "String";

SETTING → "=", RELEXPR;

CALLFUNC → "(", [RELEXPR, {",", RELEXPR}] ,")";

PRINT → "println", "(", RELEXPR, ")";

WHILE → RELEXPR, "\n", STATEMENT, "end";

IF → RELEXPR, "\n", { STATEMENT }, [ "else", "\n", STATEMENT ], "end";

FUNCTION → "function", IDENTIFIER, "("[PARAMETER], ")", "::", TYPE, "\n", STATEMENT, "end";

MATH_FUNC → MATH_FUNC_NAME, "(", RELEXPR, ")";

AVIATION_FUNC_NAME → "takeoff" | "land" | "waypoint";

TAKEOFF → "takeoff", "{", "aircraft", IDENTIFIER, "runway", IDENTIFIER, "flaps", NUMBER, "speed", NUMBER, "altitude", NUMBER, "}";

LAND → "land", "{", "aircraft", IDENTIFIER, "runway", IDENTIFIER, "flaps", NUMBER, "speed", NUMBER, "altitude", NUMBER, "}";

WAYPOINT → "waypoint", "{", "wp_name", IDENTIFIER, "speed", NUMBER, "altitude", NUMBER, "}";

MATH_FUNC_NAME → 'sqrt' | 'sin' | 'cos' | 'tan' | 'atan2' | 'log' | 'exp' | 'abs' | 'pow';

PARAMETER → IDENTIFIER, "::", TYPE, {",", IDENTIFIER, "::", TYPE};

RETURN → "return", RELEXPR;

RELEXPR → EXPRESSION, { ("==" | ">" | "<"), EXPRESSION };

EXPRESSION → TERM, { ("+" | "-" | "||" | "."), TERM };

TERM → FACTOR, { ("*" | "/" | "&&"), FACTOR };

FACTOR → (("+" | "-" | "!"), FACTOR) | NUMBER | STRING | "(", RELEXPR, ")" | IDENTIFIER, ["(", RELEXPR, {",", RELEXPR} ,")"] | ("READLN", "(", ")") | ("MATH_FUNC_NAME", "(", RELEXPR, {",", RELEXPR} ,")");

IDENTIFIER → LETTER, { LETTER | DIGIT | "_" };

NUMBER → DIGIT, { DIGIT };

LETTER → ( a | ... | z | A | ... | Z );

DIGIT → ( 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 );
```

Senta que lá vem história

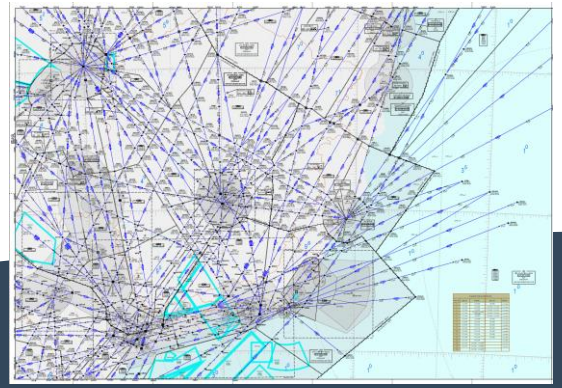
Curiosidades

Flight Management System (FMS): O FMS é um sistema computadorizado presente em aeronaves modernas, que utiliza algoritmos complexos para auxiliar na navegação, planejamento de rotas e gerenciamento de voo. Ele é programado com informações como pontos de navegação, restrições de altitude, velocidade e tempo, permitindo um voo mais eficiente e preciso.

Senta que lá vem história

Curiosidades

Algoritmos de otimização de rotas: As companhias aéreas utilizam algoritmos de otimização de rotas para planejar os voos de forma a minimizar o consumo de combustível e reduzir os tempos de voo. Esses algoritmos consideram variáveis como condições meteorológicas, restrições de espaço aéreo, tráfego e consumo de combustível, buscando encontrar a rota mais eficiente para cada voo.



Senta que lá vem história

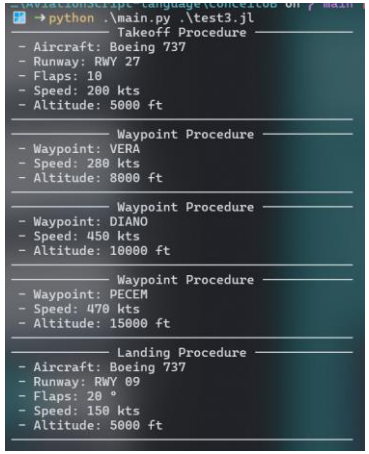
Curiosidades

Simuladores de voo: Os simuladores de voo são ferramentas essenciais para o treinamento de pilotos e também para o desenvolvimento e teste de sistemas aeronáuticos. Eles são projetados para replicar com precisão as condições de voo e são programados com modelos matemáticos complexos que simulam o comportamento da aeronave em diferentes situações.

Exemplo 1 - Definir uma rota de voo

- Utilização dos comandos TAKEOFF, WAYPOINT e LAND.
- Especificação das configurações da aeronave para decolagem e pouso.

Output:




```
TAKEOFF { "AIRCRAFT": "Boeing 737", "RUNWAY": "RWY 27", "FLAPS": 10, "SPEED": 200, "ALTITUDE": 5000 }

WAYPOINT { "WP_NAME": "VERA", "SPEED": 280, "ALTITUDE": 8000 }
WAYPOINT { "WP_NAME": "DIANO", "SPEED": 450, "ALTITUDE": 10000 }
WAYPOINT { "WP_NAME": "PECEM", "SPEED": 470, "ALTITUDE": 15000 }

LAND { "AIRCRAFT": "Boeing 737", "RUNWAY": "RWY 09", "FLAPS": 20, "SPEED": 150, "ALTITUDE": 5000 }

```

Exemplo 2 - Definir uma função para subida de altitude

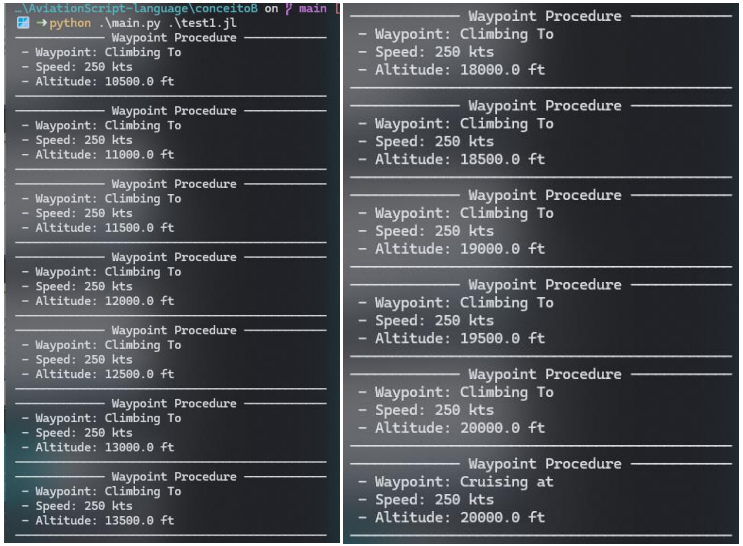
- Código exemplo para simular a subida de uma aeronave até uma altitude alvo.
- Utilização do loop WHILE para simular o processo de subida.
- Atualização da altitude e adição de waypoints.

```
altitude::Float = 10000.0
targetAltitude::Float = 20000.0
climbRate::Float = 500.0
timeToClimb::Float = (targetAltitude - altitude) / climbRate


time::Float = 0.0
t::Float = 1.0
altitudeVar::Float = altitude

while (time < timeToClimb)
    time = time + 1.0
    altitudeVar = altitudeVar + climbRate * t
    WAYPOINT { "WP_NAME": "Climbing To", "SPEED": 250, "ALTITUDE": altitudeVar }
end

WAYPOINT { "WP_NAME": "Cruising at ", "SPEED": 250, "ALTITUDE": targetAltitude }
```



Exemplo 3 - Definir uma função para calcular a distância entre dois pontos

- Código exemplo para calcular a distância entre dois pontos geográficos
- Utilização de fórmulas trigonométricas
- Retorno do valor calculado

```
function calculateDistance(lat1::Float, lon1::Float, lat2::Float, lon2::Float)::Float
  R::Int = 6371
  lat1Rad::Float = lat1 * PI / 180
  lat2Rad::Float = lat2 * PI / 180
  deltaLat::Float = (lat2 - lat1) * PI / 180
  deltaLon::Float = (lon2 - lon1) * PI / 180
  a::Float = sin(deltaLat / 2) * sin(deltaLat / 2) + cos(lat1Rad) * cos(lat2Rad) * sin(deltaLon / 2) * sin(deltaLon / 2)
  c::Float = 2 * atan2(sqrt(a), sqrt(1 - a))
  distance::Float = R * c
  return distance
end

println("Distance: ")

lat1::Float = -23.432
lon1::Float = -46.533
lat2::Float = -22.910
lon2::Float = -43.163
println(calculateDistance(lat1, lon1, lat2, lon2))
```

```
Distance:
349.34560451448414
```

Exemplo 4 - Definir uma função para converter Knots para Km/h

- Código exemplo para converter velocidade de Knots para Km/h
- Utilização de uma função e uma constante
- Retorno do valor convertido

```
function convertKnotsToKmPerHour(knots::Float)::Float
    kmPerHour::Float
    kmPerHour = knots * 1.852
    return kmPerHour
end

println("Knots to km/h: ")

knots::Float = 10.0
println(convertKnotsToKmPerHour(knots))
```

```
Knots to km/h:
18.52
```

Exemplo 5 - Definir uma função para converter Pés para Metros

- Código exemplo para converter uma medida de pés para metros
- Utilização de uma função e uma constante
- Retorno do valor convertido

```
function convertFeetToMeters(feet::Float)::Float
    meters::Float = feet * 0.3048
    return meters
end

println("Feet to meters: ")

altitudeInFeets::Float = 10.0
println(convertFeetToMeters(altitudeInFeets))
```

```
Feet to meters:
3.048
```

Exemplo 6 - Definir uma função para calcular Heading

- Código exemplo para calcular o Heading entre dois pontos geográficos
- Utilização de fórmulas trigonométricas
- Retorno do valor calculado

```
function calculateHeading(lat1::Float, lon1::Float, lat2::Float, lon2::Float)::Float
    lat1Rad::Float = lat1 * PI / 180
    lat2Rad::Float = lat2 * PI / 180
    deltaLon::Float = (lon2 - lon1) * PI / 180
    y::Float = sin(deltaLon) * cos(lat2Rad)
    x::Float = cos(lat1Rad) * sin(lat2Rad) - sin(lat1Rad) * cos(lat2Rad) * cos(deltaLon)
    heading::Float = atan2(y, x) * 180 / PI
    return y
end


println("Heading: ")
println(calculateHeading(lat1, lon1, lat2, lon2))
```

```
Heading:
0.054146682648936 71
```

Considerações finais

A linguagem AviationScript foi desenvolvida com o objetivo de facilitar a criação e execução de rotas e instruções para aeronaves. Ela oferece uma sintaxe simples e intuitiva, permitindo que os usuários definam aeronaves, rotas, waypoints e outras operações relacionadas à aviação.

Embora a AviationScript tenha sido apresentada como uma linguagem de programação em sua forma atual, existem oportunidades de expansão e aprimoramento. Por exemplo, poderia ser explorada a integração com APIs de serviços de aviação, como dados de tráfego aéreo em tempo real ou informações meteorológicas, permitindo que os usuários obtenham informações atualizadas durante a execução das rotas.

No geral, a AviationScript é uma linguagem que combina a aviação com a programação, permitindo que os entusiastas da aviação, pilotos, controladores de tráfego aéreo e desenvolvedores explorem a criação e execução de rotas e instruções para aeronaves de forma mais intuitiva e eficiente. Com sua sintaxe simples e recursos específicos para a aviação, a AviationScript abre caminho para uma maior automação e otimização dos processos envolvidos na aviação.

FIM

GitHub: **DaviReisVieira/AviationScript-language (github.com)**
LinkedIn: LinkedIn