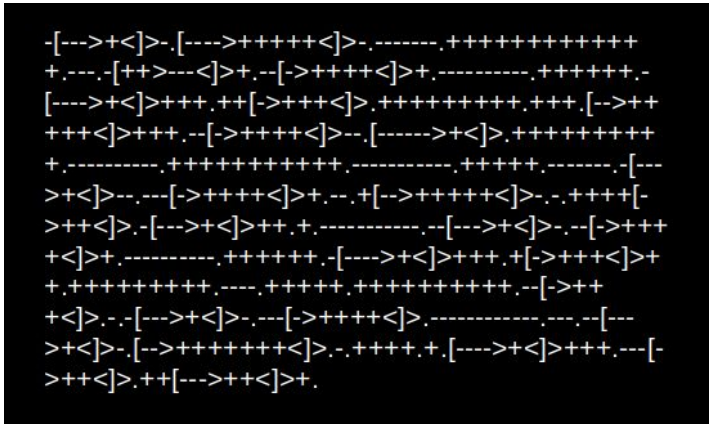
Simplified no(h)t(y)-P

Nicolas Byung Kwan Cho

Motivação

- Linguagem com propósito de confundir o usuário
- Assim como brainf**k, não é feita para uso convencional. Serve para mostrar a separação de funcionalidades entre as etapas de uma linguagem de programação:
  - Distinção entre léxico, sintático e semântico
  - Como a relação entre essas entidades geram funcionalidade para linguagem e como alterações afetam a compreensão



Um código em brainf**k

Características

- Linguagem com características compartilhadas com Python e Julia

- Inversão de tokens:
  - “"if" por "else"
  - "while" por "print"
  - "def" por "return"
- Tipagem:
  - Int por String

- Operações:
  - soma por subtração,
  - multiplicação por divisão
- Operadores lógicos:
  - "and" por "or"
- Operadores relacionais:
  - ">=" por "<="
  - ">" por "<"
  - "==" por "!="

…e vice-versa

Curiosidades

- Linguagens de programação esotéricas
  - Testam limites de design de linguagens de programação
  - Criadas usualmente como provas de conceito, software art ou como piadas e paródias
  - Usabilidade dificilmente é o foco destas linguagens
- É difícil classificar a linguagem deste projeto como esotérica, mas inspirou-se nos conceitos de parodiar outras linguagens e criar dificuldade na hora de ler e escrever

Exemplos

Condicional (if_else.npt):

```
a::String
b::String
a = 0
b = 10

else (a < b)
    while("a maior que b")
if
    while("a menor que b")
end

 #OUTPUT: a menor que b
```

String (string.npt):

```
a::Int
b::Int

a = "hello "
b = "world"
while (a.b)

#OUTPUT: hello world
```

Exemplos

Função (funcao.npt):

```
return somador(a::String,b::String)::String
    def a - b
end

a1::String
b1::String
c1::String
a1 = 2 * 2                  # 1
b1 = 2 / 2                  # 4

c1 = somador(a1,b1)
while(c1)

 #OUTPUT: 5
```

Loop (while.npt):

```
a::String
b::String
a = 0
b = 10

print(a > b)
    while(a)
    a = a - 1
end

#OUTPUT: 0 1 2 3 4 5 6 7 8 9
```