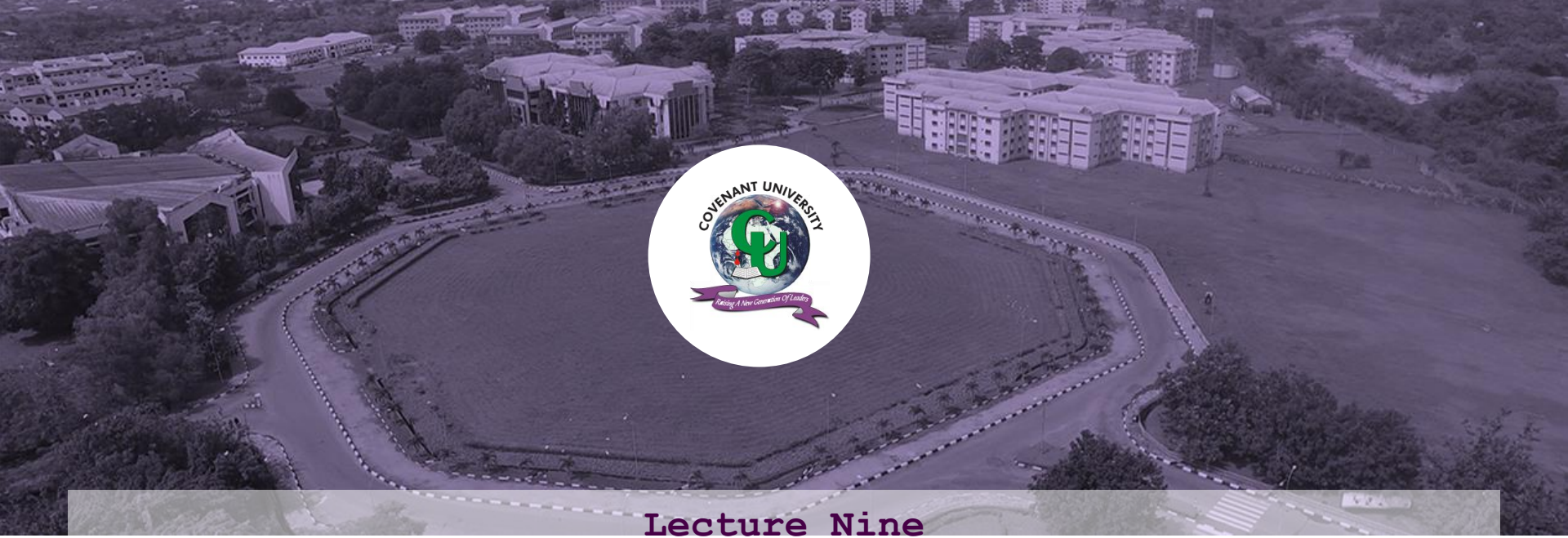


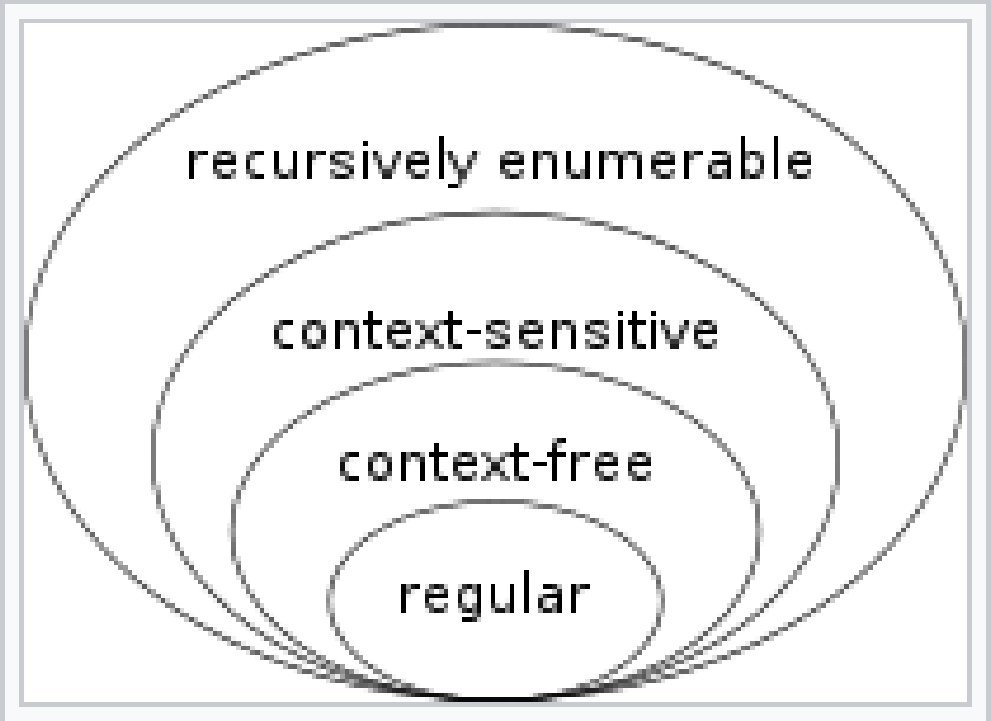
Lecture Nine

CFL, CFE, and String Derivations from CFG Production Rules

The CSC318 Team
[2023]

RECAP

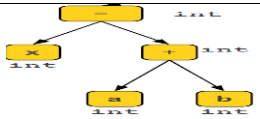
Set inclusions of formal languages described by Chomsky hierarchy.



RECAP

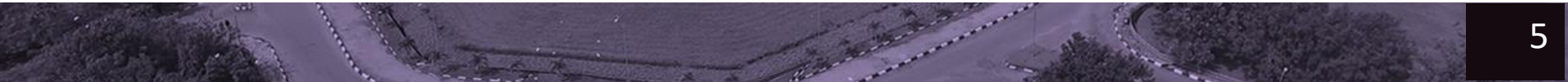
Language	Grammar	Production rules	Expression	Automata	Application in compiler / programming
Recursively-enumerable / Unrestricted	Type 0:	α --> β	Recursively-enumerable / Unrestricted	Turing Machine	Too unrestricted and too complex for use
Context-sensitive	Type 1:	γAβ --> αγβ	Context-sensitive	Linear Bounded Automata LBA	Less restricted but too complex for use
Context-free	Type 2:	A --> γ	Context-free	Push Down Automata PDA	Less restricted and less complex; syntax analysis
Regular	Type 3:	A --> a | aB	Regular	Finite Automata FA	Too restricted and too simple; lexical analysis

RECAP

Component	Input	Processing tool	Output	Example: A=B+C
Scanner	Source code	Regular grammar	Token streams	A, =, B, +, C
Parser	Tokens	Context-free grammar	Syntax / Parse tree	 Abstract Syntax tree
Semantic / Expression Analyser	Parse tree	SDT / Attribute Grammar	Attributed / Annotated tree	 Attributed Syntax tree
IC Generator	Attributed Tree	code generation algorithm and register allocation strategies	Intermediate code	t1, t2
IC Optimiser	Intermediate code	Pattern matching algorithms	Optimised TAC, Quad	t1, t2
Target code generator	Optimized TAC, Quad	Code generation algorithm & register allocation strategies	Assembly code, target code	Mov, add
Target code Optimizer	Assembly code, target code	Pattern matching algorithms	Assembly code, final target code	Mov, add

Context-Free Grammar (CFG)

- It is a formal grammar which is used to generate all possible patterns of strings in a given formal language.
- Context-free grammar G can be defined by four tuples as: G = (V, T, P, S)



Context-Free Grammar (CFG)

- **G** is the grammar, which consists of a set of the production rule.
  - It is used to generate the string of a language.
- **T** is the final set of a terminal symbol.
  - It is denoted by lower case letters.
- **V** is the final set of a non-terminal symbol.
  - It is denoted by capital letters.

Context-Free Grammar (CFG)

- **P** is a set of production rules, which is used for replacing non-terminals symbols(on the left side of the production) in a string with other terminal or non-terminal symbols(on the right side of the production).

Context-Free Grammar (CFG)

- **S** is the start symbol which is used to derive the string.
  - We can derive the string by repeatedly replacing a non-terminal by the right-hand side of the production until all non-terminal have been replaced by terminal symbols.

Context-Free Grammar (CFG)

- Example 1: Construct the CFG for the language having any number of a's over the set ∑= {a}. Then generate the string aaaaa
  - To construct CFG means to generate the production rules for the given language L or regular expression R or context-free expression C.
- Solution
  - R = a*; so that L = {(a)*}
  - P: S start symbol
  - S → aS rule 1
  - S → ε rule 2

Context-Free Grammar (CFG)

- Example 1: Solution con't. - To generate string "aaaaa"

P:	S	
S → aS	aS	rule 1
S → aS	aaS	rule 1
S → aS	aaaS	rule 1
S → aS	aaaaS	rule 1
S → aS	aaaaaS	rule 1
S → aS	aaaaaaS	rule 1
S → ε	aaaaaaε	rule 2
aaaaaa		

Context-Free Grammar (CFG)

- Example 2: Construct a CFG for the regular expression (0+1)*
  - To construct CFG means to generate the production rules for the given language L or regular expression R
- Solution
  - R = (0+1)*; so that L = {(0+1)*}
  - P: S
  - S → 0S | 1S
  - S → ε

Context-Free Grammar (CFG)

- Example 3: Construct a CFG for a language L = {wcw | where w € (a, b)*}. Derive the string “abbcbba”.
- Solution:
- For L = {wcw | where w € (a, b)*}
  - R = ((a,b)*c(a,b)*)
  - P: S start symbol
  - S → aSa rule 1
  - S → bSb rule 2
  - S → c rule 3

Context-Free Grammar (CFG)

- Example 3: Solution con't.
- To derive a string "abbcbba"
  - P: S — S — start symbol
  - S → aSa — aSa — rule 1
  - S → bSb — abSba — rule 2
  - S → bSb — abbSbba — rule 2
  - S → c — abbcbba — rule 3
  - abbcbba

Context-Free Grammar (CFG)

- Example 4: Construct a CFG for the language L = $a^n b^{2n}$ where n>=1.. Derive the string “aabbbb”.
- Solution:
- For L = $a^n b^{2n}$ where n>=1
  - L = {(a*(bb)*)}
  - C = a*(bb)*
  - P : S start symbol
  - S → aSbb | abb

Context-Free Grammar (CFG)

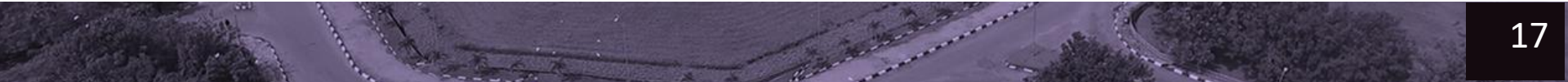
- Example 4: Solution con’t.
- To derive a string "aabbbb"
  - P : S — S — start symbol
  - S → aSabb — aSbb — rule 1
  - S → abb — aabbbb — rule 2
  - aabbbb

Derivation

- Derivation is a sequence of production rules. It is used to get the input string through these production rules.
- During parsing, we have to take two decisions.
  - decide the non-terminal which is to be replaced.
  - decide the production rule by which the non-terminal will be replaced

Derivation

- Two options to decide which non-terminal to be placed with production rule:
  - Leftmost derivation
  - Rightmost derivation



Leftmost Derivation

- In the leftmost derivation, the input is scanned and replaced with the production rule from left to right.
  - So in leftmost derivation, we read the input string from left to right.

Leftmost Derivation

- Example 1
- **Given the production rules:**
  - E = E + E
  - E = E - E
  - E = a | b
- To generate input string:
  - a – b + a
- Leftmost derivation is
  - E = E + E
  - E = E - E + E
  - E = a - E + E
  - E = a - b + E
  - E = a - b + a

Rightmost Derivation

- In rightmost derivation, the input is scanned and replaced with the production rule from right to left.
  - So in rightmost derivation, we read the input string from right to left.

Rightmost Derivation

- Example 1
- **Given the production rules:**
  - E = E + E
  - E = E - E
  - E = a | b
- To generate input string:
  - a – b + a
- Rightmost derivation is
  - E = E - E
  - E = E - E + E
  - E = E - E + a
  - E = E - b + a
  - E = a - b + a

Derivation

- Example 1:
- Derive the string "abb" for leftmost derivation and rightmost derivation using a CFG given the production rules
  - $S \rightarrow AB \mid \varepsilon$
  - $A \rightarrow aB$
  - $B \rightarrow Sb$

Derivation

- Example 1: Solution
- Using leftmost derivation for string "abb"
  - P: S — S — start symbol
  - S → AB — AB — rule 1
  - A → aB — aBB — rule 3
  - B → Sb — aSbB — rule 4
  - B → Sb — aSbSb — rule 4
  - S → ε — abSb — rule 2
  - S → ε — abb — rule 2
  - abb

Derivation

- Example 1: Solution
- Using rightmost derivation for string "abb"
  - P: S — S — start symbol
  - S → AB — AB — rule 1
  - B → Sb — ASb — rule 4
  - S → ε — Ab — rule 2
  - A → aB — ABb — rule 3
  - B → Sb — aSbb — rule 4
  - S → ε — abb — rule 2
  - abb

Derivation

- Example 2:
- Derive the string "aabbabba" for leftmost derivation and rightmost derivation using a CFG given by
  - S → aB | bA
  - A → a | aS | bAA
  - B → b | bS | aBB

Derivation

- Example 2: Solution
- Using leftmost derivation for string "abb"

P: S	S	start symbol
S → aB	aB	rule 1
B → aBB	aaBB	rule 8
B → b	aabB	rule 6
B → bS	aabbS	rule 7
S → aB	aabbaB	rule 1
B → bS	aabbabS	rule 7
S → bA	aabbabbA	rule 2
A → a	aabbabba	rule 3
aabbabba		

Derivation

- Example 2: Solution
- Using rightmost derivation for string "aabbabba"

P: S	S	start symbol
S → aB	aB	rule 1
B → aBB	aaBB	rule 8
B → bS	aaBbS	rule 7
S → bA	aaBbbA	rule 2
A → a	aaBbba	rule 3
B → bS	aabSbba	rule 7
S → bA	aabbAbba	rule 2
A → a	aabbabba	rule 3
aabbabba		

Derivation

- Example 3:
- Derive the string "00101" for leftmost derivation and rightmost derivation using a CFG given by
  - $S \rightarrow A1B$
  - $A \rightarrow 0A \mid \varepsilon$
  - $B \rightarrow 0B \mid 1B \mid \varepsilon$

Derivation

- Example 3: Solution
- Using leftmost derivation for string “00101"

P: S	S	start symbol
S → A1B	A1B	rule 1
A → 0A	0A1B	rule 2
A → 0A	00A1B	rule 2
A → ε	001B	rule 3
B → 0B	0010B	rule 4
B → 1B	00101B	rule 5
B → ε	00101	rule 6
00101		

Derivation

- Example 3: Solution
- Using rightmost derivation for string “00101"
  - P: S — S — start symbol
  - S → A1B — A1B — rule 1
  - B → 0B — A10B — rule 4
  - B → 1B — A101B — rule 5
  - B → ε — A101 — rule 6
  - A → 0A — 0A101 — rule 2
  - A → 0A — 00A101 — rule 2
  - A → ε — 00101 — rule 3
  - 00101

Exercise 1

- Consider the context-free grammar:

S -> S S + | S S * | a

1.Show how the string aa+a* can be generated by this grammar.

2.Construct a parse tree for this string.

Solution to in-class exercise

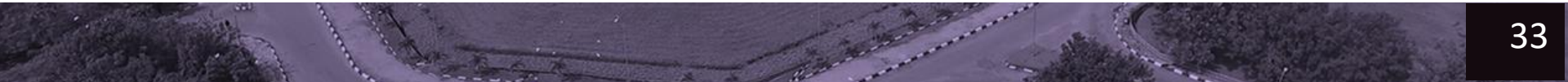
- Consider the context-free grammar production rule

(P:S-->S-|SS*|a), show how the string a-a*-a* can be generated.

P:	S	=	S
	S -->SS*	=	SS*
	S -->S-	=	S-S*
	S -->SS*	=	SS*-S*
	S -->S-	=	S-S*-S*
	S -->a	=	a-a*-a*

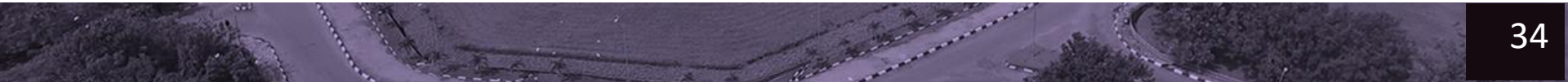
Non-English-Based Programming Languages

- Even people from countries where the local language isn't English still use English for programming.
- In fact, some of the most widely used programming languages came from non-English countries:
  - Ruby was made in Japan,
  - Jua was made in Brazil, and
  - Python was made in the Netherlands.



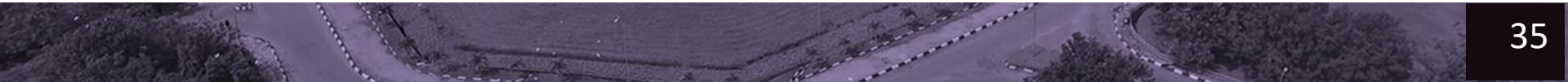
Non-English-Based Programming Languages

- For example, localized versions of Python have been created to support a variety of different languages.
- You might want to try some beginner exercises in non-English based implementations of Python, such as
  - Teuton (German),
  - Chinese Python (Chinese),
  - sawa (Javanese), or
  - Setonas (Lithuanian).



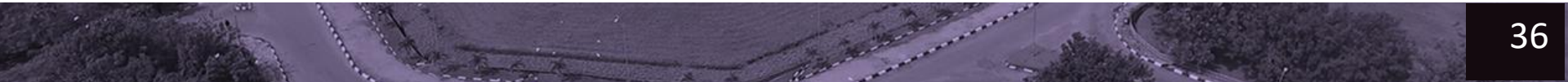
Non-English-Based Programming Languages

- English speakers are lucky because English is currently the universal programming language.
- It's considered best practice that code is written in English, and those who have a different mother tongue must learn English so they can write and read code.



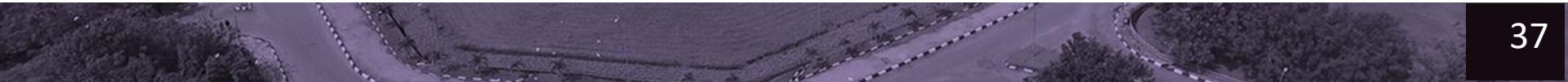
Non-English-Based Programming Languages

- While English is currently the lingua franca of the programming world, this may change as more major technologies get developed in non-English speaking nations.



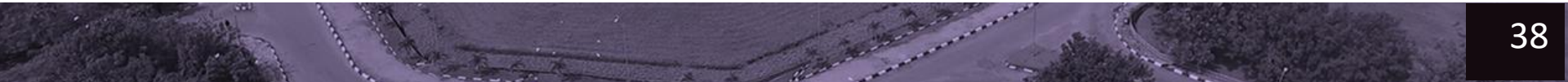
The Language of Codes : Why English is the Lingua Franca of Programming

- the total world population over 7.63 billion people and it continues to grow by the minute. **Over 1.5 billion people speak English worldwide, over 360 million native speakers**.
- Apart from being widely spoken, English is also the most commonly studied foreign language in the world.



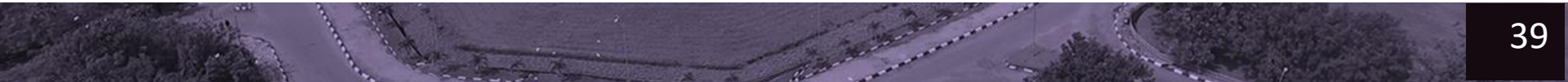
The Language of Codes : Why English is the Lingua Franca of Programming

- According to the Ethnologue: Languages of the World, a catalog of world languages, there are nearly 7,000 known languages.
- And yet, English remains dominant as the de facto standard in the global economy, especially in international business.



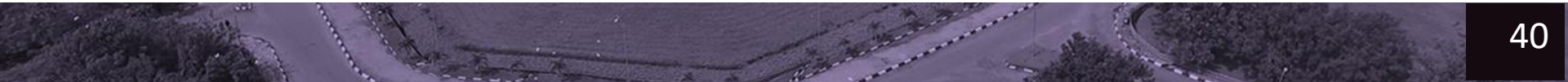
The Language of Codes : Why English is the Lingua Franca of Programming

- In the world of computer programming, English seems to be the lingua franca for coding.
  - Regardless of the original programming language, most keywords are still in English.
- Evans Data Corporation, which regularly conducts in-depth surveys of the global developer population, estimated that **there are 23 million software developers worldwide in 2018. By 2023, they are projected to be 28 million.**



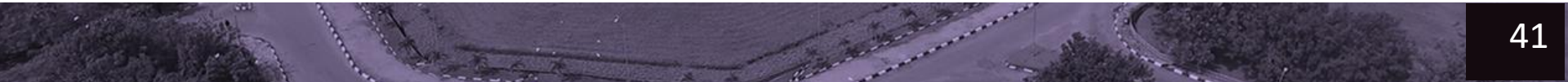
The Language of Codes : Why English is the Lingua Franca of Programming

- About 4.5 million of them are based in the U.S., with the largest number working in Silicon Valley.
  - Around 70% of these developers were born outside of the US.
- While the US currently has the largest population of software developers, India’s developer population will overtake the US by 2023.
  - The top nation for growth is in China, where it is projected to grow between 6-8% leading up to 2023.



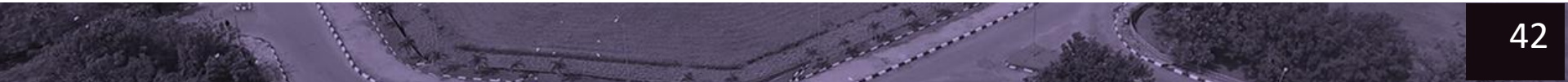
The Language of Codes : Why English is the Lingua Franca of Programming

- Recent foreign investments in technology by US companies are also spreading beyond Silicon Valley to the fastest growing tech hubs in Europe like Berlin, Dublin and Paris.
  - China’s rapid growth in advanced technology and the increasing number of Chinese-speaking developers could potentially change the status quo.



The Language of Codes : Why English is the Lingua Franca of Programming

- Most new codes are actually developed by English-speaking individuals. But not all programming codes are in English.
  - Although most keywords are written in English, comments, variable user written classes and methods are often in the programmer's own language.
- Over a third of programming language were developed in English speaking countries.
  - But some of the well-known, highly-used coding languages were developed in non-English speaking countries e.g. Switzerland (PASCAL), Denmark (PHP), Japan (Ruby), Brazil (Lua), and The Netherlands (Python).



The Language of Codes : Why English is the Lingua Franca of Programming

- Outside of learning language for the sake of learning, there are hundreds of non-English programs out there.
- Wikipedia has a comprehensive list of Non-English based programming languages.