**Vibe Coding 101 with Replit**

**Video 1: Introduction**

In this course, you will learn how to code an AI agent using Replit.

Replit provides a cloud environment that allows you to build, host, and share apps. And Replit recently added a very popular coding agent that makes generating and hosting web-based applications fast. Coding agents are changing ow we write code.

Take a problem, partition it into manageable blocks, and spend time creating a prompt describing what you want and use the model to produce a code. One module at a time.

This type of process is making me and many others much faster developers.

Note that as Replit is a cloud-hosted environment, where we are unable to present the course of DeepLearning.AI Platform, so this is a video-only course.

If you watch only the videos, you will learn key concepts behind using coding agents, as well as the process of using LLMs to write code. Sometimes you barely look at the code itself, which some now call Vibe Coding.

If outside of watching video, you have to create a user account on Replit.

**Replit**

* Provides a powerful code editor and an universal package manager. So you don’t need to struggle with installation of python and JavaScript package.
* Source control is powered by Git, but it was made as seamless as possible for our users.
* Replit offers built-in key-value stores, databases and deployment solutions.
* Now, with Replit Agent, every person with no coding experience can create apps, and deploy them in public. It will take you only a few minutes to start.

**Vibe Coding**

* Learn into letting the coding agent do most of the heavy lifting.

**Course Content**

* Start by building a search engine optimization or SEO analyzer. This is a basic app that will get us started and introduce some key concepts.
* Then you will build a head-to-head voting app which will showcase data extraction and storage.
* Throughout this course, we will learn the basic building blocks of web apps.

**Video 2: Principles of Agentic Code Development**

Tips on how to become a successful app creator with Replit agent:

1. Be precise

* Make sure to give Replit agent 1 task at a time.
* If you have a complex task in mind, break it down into smaller pieces. Otherwise, you will soon find out that adding multiple complex tasks at once can be really challenging for today’s coding agents.
* Make your prompts specific and detailed. Coding agents work best when they receive clear and unambiguous instructions.

1. Be Well-Organized

* Keep your project as tidy as possible.
  + Add features step by step.
  + Test features after each addition.
  + Don’t hesitate to roll back when something breaks.
* Always start with a fresh session for each new feature. It will be easier to go back to what was working well before.
* Do not hesitate to return to your last working version. If you keep piling up unstable code and features, most likely will end up with a mess that will hinder you from making steady progress.

1. Be patient

* Even if you are not technical, do your best to build an understanding of the app and its components as you go.
  + Replit agent explains in detail the rationale behind each action it takes, describing both the thought process and the changes it makes on the code.
  + By reading carefully the output of Replit agent, you will quickly learn everything about the app that you are creating.
* As a natural follow-up, take a moment to review suggestions before accepting them.
  + On one end, question things that don’t make sense, but on the other end, trust that Replit agent might often surprise you with an effective solution to the issue that you are facing.
* Be patient while debugging. It is part of the process when developing complex features. Even the best software developers spend most of their time debugging. Allowing it the time to debug with different strategies will make you more successful than you might expect.

**Explore the Replit Platform**

You will learn the fundamental concepts behind effective AI collaboration and key strategies when building applications with AI assistants.

**Lesson 1: Getting Started**

Vibe coding

* Definition: leaning into AI agents to rely on writing the code rather than write it ourselves.
* There is a large jump in productivity.
* A lot of people can build applications they never were able to create before.
* Can take you from “idea to app”.
* Vibe coding is limited by the developer experience (experience of developing applications or the tools, the techniques and the environments that developers use to build things.)
* Vibe coding has its “gotchas”. We are covering ways of planning and thinking logically to get around some of those limitations and fundamentally produce better results.

In this course:

* We are going to use AI to build a real, deployed apps.
* We are going to learn why tooling is important and how to make use of the tools most effectively.
* We are going to learn how to communicate with AI, because they are some fundamental skills that can help to make the most of this tooling.
* We are going to gain skills for effective building.

What Replit is:

* Replit is a workspace as the environment.
  + A unique environment, no installation, entirely in 1 tab.
  + It includes AI tools for every step of the way.
  + But more than just handling packages and languages and being zero setup, it also has everything you need to build full stack applications. That includes databases, object storage, authentication, a whole host of other tools and services.
  + Can deploy those apps in just a few clicks
* On top of Replit, we have agent (automated developer).
  + It allows you to build and configure an entire project from scratch with just your voice, generate and builds a plan for your project, and make complex, multi-step changes to your code.
* We have assistance, which is the tool for rapid edits and chats.
  + You can chat with AI about any topic, request, quick one-off edits to code and ask questions about your project.

Today, you would use Replit to scaffold out a project, get things to an MVP state, get things working, and then flipping to Assistant to really dial in that functionality and build out the rest of your project.

**5 Skills in Vibe Coding**

1. **Thinking**

* Logical
  + What is the game?
* Analytical
  + How do I play the game?
* Computational
  + What are the patterns behind the game? How we can fit a set of logic which is actually an application into a complicated problem.
  + That might lead you to do something like program a computer to enforce the rules of chess.
* Procedural
  + How do I excel at the game?
  + Programming a computer to play competitive chess.
  + What are the boundaries of the game, and soft of the edge cases that I need to account for in order to build this thing? How do I take those things and translate them into code, or translate them into natural language for AI to then implement?

1. **Frameworks**

The important thing is to start thinking about how your app works, how we are implementing solutions to these problems, and then think about solutions that already exists. Because people written a lot of code.

* You don’t know what you don’t know. At the core, try to understand what you don’t know.
* How do I do the thing I want to?
* What frameworks allow me to do that thing?
  + For example, if you want to implement a drag and drop feature into 1 of your applications, you can say “help me come up with react frameworks for implementing drag and drop into this application”, and then implement one. Interacting with AI this way is not only a great way to learn how frameworks work and how these things fit into languages, packages, and all of these other structures, but it is also a really great way to get things done faster.
* What frameworks work best with LLMs?
* If you don’t know… just ask! Like asking AI: What are common solutions to this probl em? What are some really good packages to help me to solve this problem?

1. **Checkpoints**

* Things break – this is a fact, with or without AI.
* You should use version control (or checkpoints in Replit\_ to minimize this
* We will chunk up our builds and move quickly in short sprints and utilize these checkpoints if anything breaks.

A diagram of a lamp

Description automatically generated

1. **Debugging**

* It is actually a bit boring
  + But you can make anything fun!
  + The best debugging is methodical and thorough
* Goals:
  + Understand how your app works
  + Understand where the error is
* How can you get to the root?
* How can you tell the LLM what is wrong?

1. **Context**

What do we mean when we say “context?”

Context window: the amount of tokens an LLM can process at a given time.

Context can be the prompt we provide to the LLM, but it can also be other things.

* Images
* Documentation
* Errors
* Details about your app / environment / preferences

Because LLMs might have outdated training data (or lack details of our implementation), we need to provide additional context, especially if we are doing something new or novel that hasn’t been done before, or we are working on a framework/package that the model is not an expert on.

**Getting to a MVP**

* Give AI only the information relevant to the MVP
* Start small and work our way up into something that is full-featured.
* Provide foundational context and important details when we are crafting that 1st prompt.

From there:

**Implementing new features**

* Provide context relevant to the new feature.
* Mention frameworks, provide documentation with EXPLICIT details on implementation
* Make incremental changes (checkpoint), roll them back if anything wrong happens.

**Debugging Errors**

* Figure out how things work
* Figure out what is wrong
* Figure out how to get information to the LLM to get unstuck
  + Figure out how to direct context.

A diagram of a software process

Description automatically generated

We can think about the Minimum Viable Product (MVP) or adding new features to that MVP when we are building.

We will start by prompting AI, and then we are going to test our application. We might try and break it. We might get an error and have to debug that app. But once we do and have a working version, we will get to a checkpoint. Maybe we don’t get any error and we can just go straight from testing to our checkpoint.

But the next step is to do this all over again. So it looks like we, prompt to get our MVP or feature. Then we test that out. If we hit an error, debug it, fix it. If we don’t, save it directly.

And then we move onto our next MVP or feature. And this sort of feedback loop Is largely the pattern that I follow when I am building with AI.