oriolmanzano.com

PYTHON GUIDE

24 Points to master Python language in Houdini.

oriolmanzano.com

TACTICAL GUIDE

Master Python language in 24 Steps.

By ORIOL MANZANO

| INTRODUCTION

Welcome to the Ultimate Guide to Mastering Python in Houdini! This guide is meticulously designed for enthusiasts, artists, and technical directors who wish to navigate the powerful synergy between Python programming and Houdini's robust 3D animation capabilities. Whether you're just starting or looking to enhance your skill set, we aim to equip you with the knowledge and tools necessary to automate tasks, create custom tools and interfaces, manipulate geometry, and much more. Through a blend of foundational teachings, practical examples, and advanced explorations, this guide serves as your beacon through the vast potential that Python scripting offers within the Houdini environment.

Our journey will cover essential Python basics, dive deep into the Houdini Python API, and extend into creating digital assets, managing simulations, and integrating with VEX. Each topic is designed to build upon the previous, ensuring a comprehensive understanding that is both practical and empowering. With concise explanations, actionable code examples, and valuable resources, you're set to enhance your projects and workflows significantly. As you progress, you'll find yourself unlocking new realms of creativity and efficiency, ready to tackle any challenge with confidence and skill. Let's embark on this transformative journey together, unlocking the full potential of Python in Houdini.

01. FAMILIARIZE YOURSELF WITH HOUDINI

Before delving into Python scripting, it's essential to have a strong grasp of the Houdini interface, workflows, and terminology. Understanding how to navigate the network editor, create nodes, and manipulate geometry forms the foundation upon which VEX scripting is built.

Subscribe to my youtube channel to learn the basics of houdini. Whether you're an aspiring coder or a budding VFX artist, my comprehensive video tutorials provide the perfect platform to hone your skills and unlock your full potential.

02. Understanding Python Syntax

Python is renowned for its readability and ease of use, making it an excellent choice for scripting in Houdini. Basic syntax includes defining variables, writing function definitions, and control flow statements.

For instance, to define a simple function in Python that prints the name of a node:

oriolmanzano.com python

```python
def print_node_name(node):
    print(node.name())
```

In Houdini, you would use this function by passing a node object from the hou module, which might look like this:

oriolmanzano.com python

```python
node = hou.node('/obj/geo1')
print_node_name(node)
```

For more on Python syntax, the official Python documentation is an invaluable resource.

03. ACCESING PARAMETERS

Accessing and modifying node parameters is a fundamental operation when scripting in Houdini. The **hou** module provides methods to interact with these parameters. Here's how you can get and set the value of a parameter:

oriolmanzano.com python

```python
node = hou.node('/obj/geo1')
# Get the value of a parameter
translate = node.parm('tx').eval()
print(f"Current X translation: {translate}")

# Set the value of a parameter
node.parm('tx').set(5.0)
```

This script changes the X translation of an object named 'geo1' to 5.0 units.

04. CREATING AND EDITING NODES

Automating the creation and configuration of nodes is a powerful aspect of Houdini's Python API. For example, creating a sphere and then connecting it to a transform node can be done like this:

oriolmanzano.com python

```python
# Create a geometry node in the object level
geo = hou.node('/obj').createNode('geo', 'MyGeo')

# Inside the geometry node, create a sphere and a
transform node
sphere = geo.createNode('sphere')
transform = geo.createNode('xform')

# Connect the sphere to the transform node
sphere.setOutput(0, transform)

# Cook the geo node to update the scene
geo.cook()
```

This script showcases how to programmatically add nodes and define their connections, a common task in Houdini scripting.

05. WORKING WITH GEOMETRY ATTRIBUTES

Direct manipulation of geometry and attributes is a bit more advanced but unlocks a lot of procedural capabilities in Houdini. For instance, adding a custom attribute to every point in a geometry can be achieved as follows:

oriolmanzano.com python

```python
# Access the geometry of an existing node
geo = hou.node('/obj/geo1').geometry()

# Add a float attribute named 'myAttr' to all points
attr = geo.addAttrib(hou.attribType.Point, "myAttr",
0.0)

# Iterate through points and set the attribute's value
for point in geo.points():
    point.setAttribValue(attr, 1.0)
```

This script introduces a new point attribute and assigns a value to it for every point in the geometry.

06. INTEGRATION WITH VEX

While VEX is the native language for writing shaders, custom nodes, and effects in Houdini, Python can call VEX functions and use VEX code snippets. This interaction is particularly useful for tasks that are more efficiently executed in VEX. However, integrating Python with VEX requires an understanding of both languages and how they can interoperate within Houdini's context. The Houdini documentation and forums are great resources for exploring specific examples of Python and VEX integration.

07. DYNAMICS AND SIMULATIONS

Houdini excels at dynamics and simulations, areas where Python scripts can control and automate complex systems. For instance, adjusting simulation parameters based on feedback or external data sources:

oriolmanzano.com python

```python
# Access a DOP network and a specific simulation parameter
dopnet = hou.node('/obj/AutoDopNetwork')
gravity = dopnet.parm('gravity')

# Dynamically adjust gravity based on external conditions
gravity.set(-9.8 * 1.5)  # Increase gravity by 50%
```

This script exemplifies how to script simulation parameters, a technique that can be expanded to control virtually any aspect of Houdini's dynamic systems.

08. DEBUGGING AND ERROR HANDLING

When scripting in Houdini, you'll inevitably encounter errors. Proper error handling and debugging are essential for creating robust scripts. Python's try-except block is a fundamental tool for this:

oriolmanzano.com python

```python
try:
    # Attempt to access a node that might not exist
    node = hou.node('/obj/nonexistent_node')
    node.parm('tx').set(10)
except AttributeError:
    print("Failed to find the node or parameter.")
```

This example attempts to set a parameter on a potentially non-existent node, catching the **AttributeError** that would be raised if either the node or the parameter does not exist.

09. BATCH PROCESSING

Batch processing allows you to automate repetitive tasks over multiple files or data sets. This can save considerable time, especially for tasks like rendering or geometry processing:

oriolmanzano.com python

```python
import os

# Directory containing .hip files
hip_files_dir = "/path/to/hip/files"
hip_files = [f for f in os.listdir(hip_files_dir) if
f.endswith('.hip')]

for hip_file in hip_files:
    hip_path = os.path.join(hip_files_dir, hip_file)
    # Use hou.hipFile to work with .hip files
    try:
        hou.hipFile.load(hip_path,
suppress_save_prompt=True)
        # Perform some operations here, e.g., rendering
        print(f"Processed {hip_file}")
    except hou.LoadWarning as lw:
        print(f"Warning while loading {hip_file}: {lw}")
```

This script iterates through a directory of Houdini project files, opens each one, and performs operations on them, illustrating batch processing's power.

10. WORKING WITH EXTERNAL LIBRARIES

Python's extensive collection of libraries can be leveraged in Houdini to extend its capabilities. For instance, you might use numpy for complex mathematical operations:

oriolmanzano.com python

```python
import numpy as np

# Create an array of point positions
points = np.array([[0, 0, 0], [1, 1, 1], [2, 2, 2]])

# Perform an operation, like scaling
scaled_points = points * 2

# Now, you could use these scaled points to modify
geometry in Houdini
```

This snippet shows how to use numpy for manipulating point data, which could then be applied to geometry within Houdini.

11. CREATING DIGITAL ASSETS WITH PYTHON

Digital assets in Houdini (HDAs) can encapsulate complex setups for reuse. Python can automate the creation and configuration of HDAs:

oriolmanzano.com python

```python
# Create a new geometry node
geo_node = hou.node('/obj').createNode('geo',
'MyGeoAsset')

# Define the node's parameters or setup as needed
# ...

# Create a digital asset from this node
definition =
geo_node.createDigitalAsset(name="my_geo_asset",

hda_file_name="/path/to/save/my_geo_asset.hda",
                              description="A custom geo asset",
                              version="1.0")

# Now you have a digital asset that can be shared or
reused
```

This example demonstrates how to programmatically create a digital asset from a geometry node, setting the foundation for automating complex asset creation.

12. AUTOMATING UI ELEMENTS

For tools and scripts that require user interaction, automating UI creation is crucial. Houdini's Python API allows for dynamic UI creation:

oriolmanzano.com python

```python
import hou

def my_custom_ui():
    # Create a dialog window with user inputs
    dialog = hou.ui.readMultiInput("Enter parameters for the operation:", ["Parameter 1", "Parameter 2"])

    if dialog[0] == hou.ui.DialogResponseType.Accepted:
        print(f"User input: {dialog[1]}")
    else:
        print("Operation canceled by user.")

# Invoke the UI
my_custom_ui()
```

This script shows how to prompt the user with a simple dialog window, collect input, and handle the response, illustrating how to integrate user feedback into your scripts.

13. CUSTOM OPERATORS WITH PYTHON

Houdini allows the creation of custom operators or Houdini Digital Assets (HDAs) using Python, which can encapsulate complex behaviors and processes:

oriolmanzano.com python

```
# Define a new Python SOP
class MyCustomSOP(hou.SopNode):
    def __init__(self, *args, **kwargs):
        super(MyCustomSOP, self).__init__(*args,
**kwargs)

    def cook(self, geometry):
        # Your cooking logic here
        pass

# Register the operator
hou.sopNodeTypeCategory().addNodeType('MyCusto
mSOP', MyCustomSOP, icon="SOP_torus")
```

This code defines a skeleton for a custom SOP (Surface Operator) that does nothing at the moment but provides a template for implementing your cooking logic.

14. EVENT-DRIVEN SCRIPTING

Houdini scripts can respond to various events, such as changes in the node network or parameter updates, enabling dynamic and responsive tools:

oriolmanzano.com python

```python
def parameter_changed_callback(event_type, **kwargs):
    if event_type == hou.nodeEventType.ParmValueChanged:
        node = kwargs["node"]
        changed_parm = kwargs["parm"]
        print(f"Parameter {changed_parm.name()} on node {node.name()} was changed.")

# Register the callback
hou.nodeCallbacks().addCallback(parameter_changed_callback)
```

This example shows how to set up a callback that prints a message whenever any parameter value changes in the node network, illustrating the basics of event-driven scripting in Houdini.

15. INTERFACING WITH EXTERNAL APPLICATIONS

Python scripts in Houdini can communicate with other applications, allowing for workflows that span multiple tools. This can be done through various means, including command line tools, web APIs, or file interchange formats:

oriolmanzano.com python

```python
import subprocess

# Example: Call an external command-line tool
subprocess.run(["my_external_tool", "-arg", "value"])

# Example: Send data to a web API (simplified)
import requests
response = requests.post("http://myapi.com/data",
json={"my": "data"})
print(response.text)
```

These snippets demonstrate calling an external command-line tool and sending data to a web API, respectively, showcasing how Houdini can be part of a larger pipeline.

16. SCRIPTING FOR DATA VISUALIZATION

Houdini's powerful geometry and rendering capabilities make it an excellent tool for data visualization. Python scripts can generate geometric representations from data:

oriolmanzano.com python

```python
import random

geo = hou.node('/obj').createNode('geo',
'DataVisualization')
points = [geo.createNode('add', 'Point_{}'.format(i)) for
i in range(10)]

for point in points:
    # Set random positions for each point
    point.parmTuple('pt0').set((random.uniform(-1, 1),
random.uniform(-1, 1), random.uniform(-1, 1)))

# Optionally, add more nodes to visualize
connections, surfaces, etc.
```

This code creates a series of points with random positions, serving as a basic example of data visualization.

17. CUSTOM UIs FOR TOOLS AND SCRIPTS

Building on the concept of automating UI elements, you can create more complex and interactive user interfaces for your tools using PySide2 or PyQt:

oriolmanzano.com python

```python
from PySide2 import QtWidgets

class MyToolUI(QtWidgets.QWidget):
    def __init__(self, parent=None):
        super(MyToolUI, self).__init__(parent)
        self.layout = QtWidgets.QVBoxLayout(self)
        self.label = QtWidgets.QLabel("My Custom Tool", self)
        self.layout.addWidget(self.label)

# Display the UI
app = QtWidgets.QApplication([])
ui = MyToolUI()
ui.show()
app.exec_()
```

This example demonstrates creating a basic UI with a label, illustrating the first step toward more sophisticated tool interfaces.

18. PERFORMANCE OPTIMIZATION IN SCRIPTS

Efficiently written Python scripts are crucial for maintaining performance, especially when dealing with large datasets or complex operations in Houdini:

oriolmanzano.com python

```python
# Example: Use hou.Geometry.freeze() for heavy operations
geo = hou.node('/obj/geo1').geometry().freeze()
# Perform operations on geo without triggering updates in the scene...
```

The use of **.freeze()** on a geometry object allows for operations to be performed without updating the scene after each change, illustrating a simple optimization technique.

19. LEVERAGING NETWORK ARCHITECTURE

Understanding and manipulating Houdini's node network via Python provides a powerful means to automate and customize your workflow:

oriolmanzano.com python

```python
# Example: Create a series of connected nodes
node = hou.node('/obj').createNode('geo',
'DynamicGeometry')
box = node.createNode('box')
transform = node.createNode('xform')
box.setNextInput(transform)

# Automatically layout nodes for clarity
node.layoutChildren()
```

This script demonstrates creating a geometry node containing a box and a transform node, showcasing how to programmatically manipulate the node network.

20. SCRIPTING FOR PROCEDURAL MODELING

Procedural modeling is one of Houdini's core strengths. Python can be used to create or modify procedural models, offering a scriptable approach to complex geometry creation:

oriolmanzano.com python

```python
# Example: Create a procedural staircase
geo = hou.node('/obj').createNode('geo', 'Staircase')
copy = geo.createNode('copy')
box = geo.createNode('box')
transform = geo.createNode('xform')

transform.parm('ty').set(0.5)
copy.setInput(0, box)
copy.setInput(1, transform)
copy.parm('ncy').set(10)  # Number of copies

geo.layoutChildren()
```

This example illustrates how to create a simple procedural staircase by copying a box geometry with transformations, emphasizing Python's role in procedural modeling.

21. INTEGRATING WITH MACHINE LEARNING MODELS

With the growing accessibility of machine learning models, integrating them into Houdini workflows opens up innovative possibilities for automation and creativity:

oriolmanzano.com python

```python
# Pseudocode: Use a machine learning model to generate geometry based on input parameters
import hou
import my_ml_model

def generate_geometry_with_ml(input_params):
    # Generate model input from parameters
    model_input = my_ml_model.prepare_input(input_params)

    # Predict geometry data with the model
    geometry_data = my_ml_model.predict(model_input)

    # Convert predicted data into Houdini geometry
    geo = hou.node('/obj').createNode('geo', 'GeneratedGeo')
    # Assuming geometry_data contains points, create them in Houdini
    for point_data in geometry_data['points']:
        geo.createPoint(point_data['position'])

    return geo

# Example usage
input_params = {'complexity': 5, 'symmetry': True}
generated_geo = generate_geometry_with_ml(input_params)
```

This pseudocode outlines how a machine learning model could be used to generate geometry based on input parameters, illustrating the potential for advanced automation and creative expression.

22. CUSTOM RENDER PROCEDURES

Python can also control rendering processes, allowing for customized rendering pipelines, pre/post-processing, or automation of rendering tasks:

oriolmanzano.com python

```python
# Example: Automate rendering of multiple cameras
render_node = hou.node('/out/mantra')
cameras = hou.node('/obj').glob('cam*')  # Assumes cameras are named cam1, cam2, etc.

for cam in cameras:
    render_node.parm('camera').set(cam.path())
    render_node.render()
```

This script automates rendering from multiple cameras, showcasing how Python can streamline rendering workflows in Houdini.

23. SIMULATION CONTROL AND ANALYSIS

Simulations are a critical component of Houdini's toolset. Python scripting can be used to control simulation parameters dynamically or analyze simulation results for further processing:

oriolmanzano.com python

```python
# Example: Adjust simulation parameters based on initial results
sim_node = hou.node('/obj/AutoDopNetwork')
initial_velocity = sim_node.parm('velocity').eval()

if initial_velocity < 10:
    sim_node.parm('gravity').set(-9.81)  # Adjust gravity for low initial velocity
else:
    sim_node.parm('gravity').set(-1.62)  # Lunar gravity for high initial velocity

# Re-run simulation with adjusted parameters
sim_node.simulate()
```

This example dynamically adjusts simulation parameters based on initial conditions, demonstrating Python's potential to refine simulations through scripting.

24. ENHANCING ASSET MANAGEMENT

In large projects or studio pipelines, efficient asset management becomes crucial. Python scripts can automate asset tracking, versioning, and updates:

oriolmanzano.com python

```python
# Example: Batch update asset versions
asset_nodes = hou.node('/obj').glob('asset_*')
latest_version = 'v1.2'

for asset in asset_nodes:
    if asset.type().definition().version() < latest_version:
        # Assuming an updateAssetVersion function
exists
        updateAssetVersion(asset, latest_version)
```

This script identifies asset nodes needing updates and applies the latest version, illustrating Python's role in

| ABOUT THE AUTHOR

I am **Oriol Manzano**, a visionary at the convergence of computer science and visual arts, driven by a passion for innovation and creativity. My journey began in Barcelona, where I developed a fascination with technology's transformative potential. Pursuing my education at Epitech University in Toulouse and Paris, I explored the intersections of 3D design, artificial intelligence, and machine learning. Collaborating with esteemed companies like Axis Studios, Virtuos, and Technicolor, I've contributed to AAA gaming titles such as "Sackboy: A Big Adventure" for the PlayStation 5 and cinematic experiences like "Transformers 4" and "Ant-Man," crafting immersive visual effects and captivating storytelling. Beyond work, I'm committed to pushing the boundaries of technology and art, championing creativity's transformative power, and leaving a lasting impact in the ever-evolving landscape of computer science and visual arts.

oriolmanzano.com

OR · 75