Python Syntax Cheat Sheet for Product Managers

Variables & Data Types

Variable Assignment

python

```python
# Creating variables (like labeling data containers)
user_count = 1500
conversion_rate = 0.045
product_name = "Analytics Dashboard"
is_premium_user = True
```

Basic Data Types

python

```python
# Numbers
monthly_users = 50000          # Integer
churn_rate = 12.5              # Float

# Text
feedback = "Great app!"        # String
customer_id = "USER_12345"     # String

# True/False values
feature_enabled = True         # Boolean
test_running = False           # Boolean
```

Type Checking & Conversion

python

```python
# Check what type of data you have
type(user_count)               # Returns: <class 'int'>
type(conversion_rate)          # Returns: <class 'float'>

# Convert between types
str(1500)                      # "1500" (number to text)
int("1500")                    # 1500 (text to number)
float(1500)                    # 1500.0 (integer to decimal)
```

Basic Operations

Math Operations

python

```python
# Basic calculations
total_revenue = 10000 + 5000   # Addition: 15000
profit_margin = 15000 - 8000   # Subtraction: 7000
quarterly_revenue = 5000 * 3   # Multiplication: 15000
avg_deal_size = 15000 / 50     # Division: 300.0
users_per_cohort = 1000 // 4   # Integer division: 250
user_id_remainder = 1001 % 10  # Remainder: 1

# Useful for calculations
engagement_score = 85 ** 2     # Power: 7225
import math
sqrt_users = math.sqrt(10000)  # Square root: 100.0
```

String Operations

python

```python
# Working with text
product_name = "Mobile App"
company = "TechCorp"

# Combining text
full_title = product_name + " by " + company  # "Mobile App by TechCorp"
welcome_msg = f"Welcome to {product_name}!"   # f-string formatting

# Changing text case
product_name.upper()          # "MOBILE APP"
product_name.lower()          # "mobile app"
company.title()               # "Techcorp"

# Text information
len(product_name)             # Length: 10
"App" in product_name         # Check if text contains: True
```

Comparison Operations

python

```python
# Comparing values (returns True or False)
user_count > 1000             # Greater than
conversion_rate < 0.05        # Less than
churn_rate >= 10.0           # Greater than or equal
retention_rate <= 85.0       # Less than or equal
status == "active"           # Equal to
plan_type != "free"          # Not equal to
```

Collections (Lists & Dictionaries)

Lists (Ordered Collections)

python

```python
# Creating lists
user_segments = ["New", "Active", "At Risk", "Churned"]
monthly_signups = [120, 150, 98, 200, 175]

# Accessing list items (starts counting from 0)
first_segment = user_segments[0]        # "New"
last_signup = monthly_signups[-1]       # 175 (last item)

# Adding to lists
user_segments.append("VIP")             # Add to end
monthly_signups.insert(0, 110)         # Insert at position 0

# List information
len(user_segments)                      # Number of items: 5
"Active" in user_segments               # Check if item exists: True

# Working with lists
total_signups = sum(monthly_signups)    # Add all numbers
max_signups = max(monthly_signups)      # Highest number
user_segments.sort()                    # Sort alphabetically
```

Dictionaries (Key-Value Pairs)

python

```python
# Creating dictionaries (like user profiles)
user_profile = {
    "user_id": "U12345",
    "name": "Sarah Chen",
    "plan": "Premium",
    "signup_date": "2024-01-15",
    "active": True
}

# Accessing dictionary values
user_name = user_profile["name"]              # "Sarah Chen"
user_plan = user_profile.get("plan")          # "Premium"
user_country = user_profile.get("country", "Unknown")  # Default if missing

# Adding/updating values
user_profile["last_login"] = "2024-06-03"     # Add new key
user_profile["plan"] = "Enterprise"           # Update existing key

# Dictionary information
user_profile.keys()                           # All keys
user_profile.values()                         # All values
"name" in user_profile                        # Check if key exists: True
```

Control Structures

If Statements (Decision Making)

python

```python
# Basic if statement
if user_count > 10000:
    print("High user base!")

# If-else
if conversion_rate > 0.05:
    print("Good conversion")
else:
    print("Needs improvement")

# Multiple conditions
if churn_rate < 5:
    risk_level = "Low"
elif churn_rate < 15:
    risk_level = "Medium"
else:
    risk_level = "High"

# Combining conditions
if user_count > 1000 and conversion_rate > 0.03:
    print("Healthy metrics")

if plan == "free" or usage_days < 7:
    print("Show upgrade prompt")
```

Loops (Repeating Actions)

For Loops (Process Each Item)

python

```python
# Loop through a list
user_segments = ["New", "Active", "At Risk"]
for segment in user_segments:
    print(f"Processing {segment} users")

# Loop through numbers
for month in range(1, 13):  # 1 to 12
    print(f"Month {month}")

# Loop through dictionary
user_data = {"name": "John", "plan": "Pro", "active": True}
for key, value in user_data.items():
    print(f"{key}: {value}")
```

While Loops (Repeat Until Condition)

python

```python
# Keep trying until success
attempts = 0
while attempts < 3:
    print(f"API call attempt {attempts + 1}")
    attempts += 1

# Process until empty
pending_tasks = ["Review A/B test", "Update roadmap", "Analyze churn"]
while pending_tasks:
    current_task = pending_tasks.pop(0)
    print(f"Completing: {current_task}")
```

Functions (Reusable Code Blocks)

Basic Functions

python

```python
# Simple function
def calculate_churn_rate(churned_users, total_users):
    return (churned_users / total_users) * 100

# Function with default parameter
def send_notification(message, urgent=False):
    if urgent:
        print(f"URGENT: {message}")
    else:
        print(f"Info: {message}")

# Using functions
churn = calculate_churn_rate(50, 1000)  # Returns: 5.0
send_notification("New feature launched")
send_notification("Server down", urgent=True)
```

Functions with Multiple Returns

python

```python
def analyze_conversion_funnel(visitors, signups, purchases):
    signup_rate = (signups / visitors) * 100
    purchase_rate = (purchases / signups) * 100
    overall_rate = (purchases / visitors) * 100

    return signup_rate, purchase_rate, overall_rate

# Using the function
sign_rate, purch_rate, overall = analyze_conversion_funnel(10000, 500, 50)
```

Input/Output Operations

Console Input/Output

python

```python
# Getting user input
user_name = input("Enter your name: ")
age = int(input("Enter your age: "))

# Displaying output
print("Hello, Product Manager!")
print(f"User {user_name} is {age} years old")

# Formatting output
revenue = 125000.75
print(f"Revenue: ${revenue:,.2f}")  # "Revenue: $125,000.75"
```

File Operations

python

```python
# Reading from a file
with open("user_data.txt", "r") as file:
    content = file.read()
    lines = file.readlines()

# Writing to a file
with open("report.txt", "w") as file:
    file.write("Monthly Report\n")
    file.write(f"Total Users: {user_count}\n")

# Appending to a file
with open("log.txt", "a") as file:
    file.write("New entry added\n")
```

Common Patterns for Product Managers

Data Processing Pattern

python

```python
# Process a list of user data
users = [user1, user2, user3]  # List of user dictionaries
active_users = []

for user in users:
    if user["status"] == "active":
        active_users.append(user)

print(f"Found {len(active_users)} active users")
```

Counting Pattern

python

```python
# Count occurrences
feedback_categories = ["bug", "feature", "bug", "praise", "feature", "bug"]
category_counts = {}

for category in feedback_categories:
    if category in category_counts:
        category_counts[category] += 1
    else:
        category_counts[category] = 1

# Result: {"bug": 3, "feature": 2, "praise": 1}
```

Finding Maximum/Minimum Pattern

python

```python
# Find best performing feature
feature_usage = [
    {"name": "Dashboard", "usage": 85},
    {"name": "Reports", "usage": 67},
    {"name": "Analytics", "usage": 92}
]

best_feature = feature_usage[0]
for feature in feature_usage:
    if feature["usage"] > best_feature["usage"]:
        best_feature = feature

print(f"Most used feature: {best_feature['name']}")
```

Validation Pattern

python

```python
# Validate user input
def validate_email(email):
    if "@" in email and "." in email:
        return True
    return False

def validate_signup_data(user_data):
    errors = []

    if not user_data.get("name"):
        errors.append("Name is required")

    if not validate_email(user_data.get("email", "")):
        errors.append("Valid email is required")

    return errors

# Usage
signup_data = {"name": "John", "email": "john@example.com"}
validation_errors = validate_signup_data(signup_data)
if validation_errors:
    print("Signup failed:", validation_errors)
else:
    print("Signup successful!")
```

Quick Tips

- **Variable names**: Use descriptive names like `conversion_rate` instead of `cr`
- **Indentation**: Python uses spaces (4 spaces per level) to group code
- **Comments**: Use `#` for single-line comments, `"""text"""` for multi-line
- **Case sensitivity**: `User` and `user` are different variables
- **Index counting**: Lists start counting from 0, not 1