python_basics

February 6, 2026

1 Python Programming Fundamentals

This notebook contains exercises to test your understanding of basic Python concepts. Complete each code cell according to the instructions. When you run the cell, it will tell you if your answer is correct.

1.1 Question 1: Variables and Data Types

Create variables of different types: - Create an integer variable named `age` with value 25 - Create a float variable named `height` with value 1.75 - Create a string variable named `name` with value "Alice" - Create a boolean variable named `is_student` with value True

[1]:
```
# Your code here:
age = 25
height = 1.75
name = "Alice"
is_student = True

# Don't modify the test code below
assert isinstance(age, int) and age == 25
assert isinstance(height, float) and height == 1.75
assert isinstance(name, str) and name == "Alice"
assert isinstance(is_student, bool) and is_student == True
print("Question 1 passed!")
```

```
Question 1 passed!
```

1.2 Question 2: Lists and Indexing

Create a list named `numbers` containing numbers 1 through 5, then: - Print the second element - Print the last element using negative indexing - Print a slice containing the second and third elements

[3]:
```
# Your code here:
numbers = [1, 2, 3, 4, 5]
print(numbers[-1])
print(numbers[0:3])

# Test code
```

```
assert numbers == [1, 2, 3, 4, 5]
assert len(numbers) == 5
print("Question 2 passed!")
```

```
5
[1, 2, 3]
Question 2 passed!
```

1.3 Question 3: Functions

Write a function called `calculate_average` that: - Takes a list of numbers as input - Returns the average (mean) of those numbers - Handles empty lists by returning 0

[4]:
```
# Your code here:
def calculate_average(nums):
    if len(nums) == 0:
        return 0
    return sum(nums) / len(nums)

# Test code
test_cases = [
    ([1, 2, 3, 4, 5], 3.0),
    ([10], 10.0),
    ([], 0)
]

for test_input, expected in test_cases:
    result = calculate_average(test_input)
    assert result == expected, f"Failed: input {test_input}, got {result},␣
 ↪expected {expected}"
print("Question 3 passed!")
```

```
Question 3 passed!
```

1.4 Question 4: Dictionaries

Create a function called `count_words` that: - Takes a string as input - Returns a dictionary where keys are words and values are the number of times each word appears - Ignores case (converts all words to lowercase) - Removes punctuation

[12]:
```
# Your code here:
def count_words(text):
    word_count = {}
    for word in text.lower().split():
        if word in word_count:
            word_count[word] += 1
        else:
            word_count[word] = 1
    return word_count
```

```
# Test code
test_string = "Hello World Hello Python hello WORLD"
result = count_words(test_string)
assert result["hello"] == 3
assert result["world"] == 2
assert result["python"] == 1
print("Question 4 passed!")
```

```
Question 4 passed!
```

1.5 Question 5: Control Flow

Write a function called `quantum_software` that: - Takes a number n as input - Returns "Quantum" if the number is divisible by 3 - Returns "Software" if the number is divisible by 5 - Returns "QuantumSoftware" if the number is divisible by both 3 and 5 - Returns the number as a string if none of the above conditions are met

[13]:
```
# Your code here:
def quantum_software(num):
    if num % 3 == 0 and num % 5 == 0:
        return "QuantumSoftware"
    elif num % 3 == 0:
        return "Quantum"
    elif num % 5 == 0:
        return "Software"
    else:
        return str(num)

# Test code
test_cases = [
    (15, "QuantumSoftware"),
    (3, "Quantum"),
    (5, "Software"),
    (7, "7")
]

for num, expected in test_cases:
    result = quantum_software(num)
    assert result == expected, f"Failed: input {num}, got {result}, expected␣
 ↪{expected}"
print("Question 5 passed!")
```

```
Question 5 passed!
```

1.6 Question 6: Classes

Create a class called `BankAccount` that: - Has attributes for `account_holder` (string) and `balance` (float) - Has an `__init__` method that sets these attributes - Has a `deposit` method that adds

money to the balance - Has a `withdraw` method that: - Subtracts money from the balance if sufficient funds exist - Raises a ValueError if insufficient funds - Has a `get_balance` method that returns current balance

[16]:
```python
# Your code here:
class BankAccount:
    def __init__(self, account_holder, balance):
        self.account_holder = account_holder
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount > self.balance:
            raise ValueError("Insufficient funds")
        self.balance -= amount

    def get_balance(self):
        return self.balance

# Test code
account = BankAccount("John Doe", 100.0)
assert account.account_holder == "John Doe"
assert account.get_balance() == 100.0

account.deposit(50.0)
assert account.get_balance() == 150.0

account.withdraw(30.0)
assert account.get_balance() == 120.0

try:
    account.withdraw(200.0)
    assert False, "Should have raised ValueError"
except ValueError:
    pass

print("Question 6 passed!")
```

```
Question 6 passed!
```