Loop and Debug

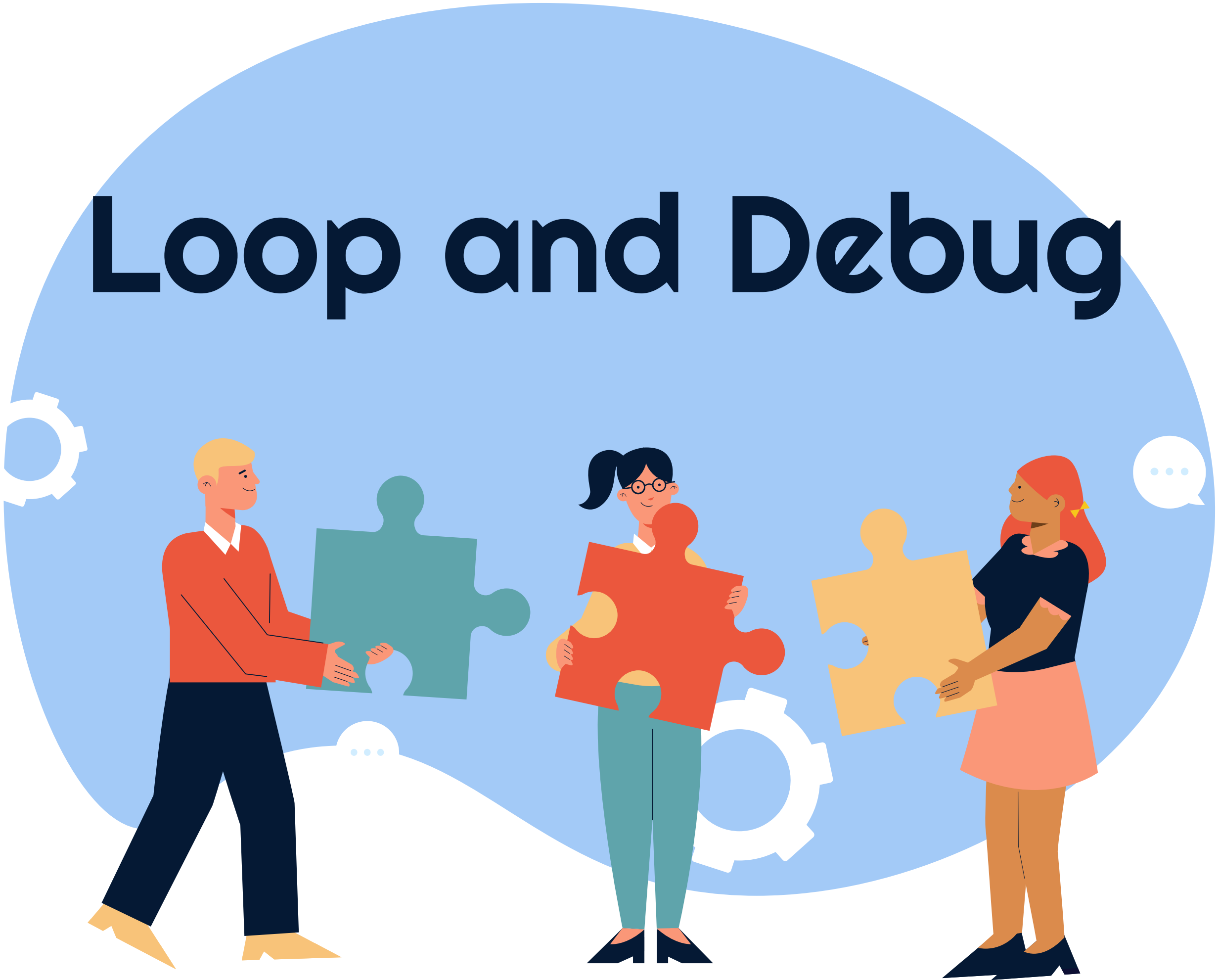


TABLE OF CONTENTS

01

Python range() and for loop

02

For loop

03

While loop

04

Debugging techniques

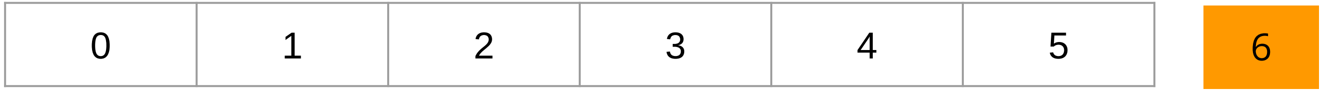
01

range() function

What is a range() ?

Python built-in range() function generates the **integer numbers between the given start integer to the stop integer**

```
range(6)
```



0	1	2	3	4	5	6

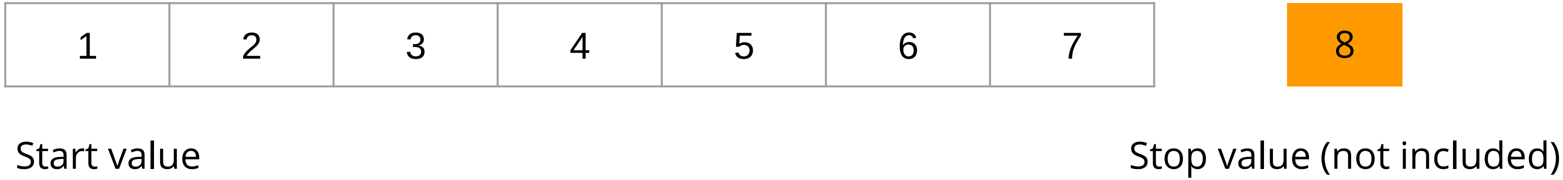
Stop value (not included)

Start value (default is 0)

range(start_value, stop_value)

range() function use the first argument as the starting point for the list, the second argument will be the stop value (exclusive)

```
range(1, 8)
```



1	2	3	4	5	6	7		8

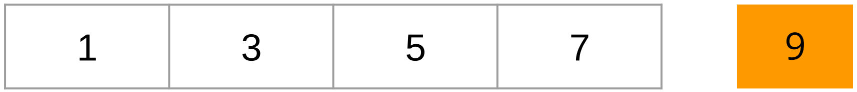
Start value

Stop value (not included)

range(start_value, stop_value, step)

The third argument specify the step the next number will be generated, if not provided, the default will be 1

```
range(1, 9, 2)
```

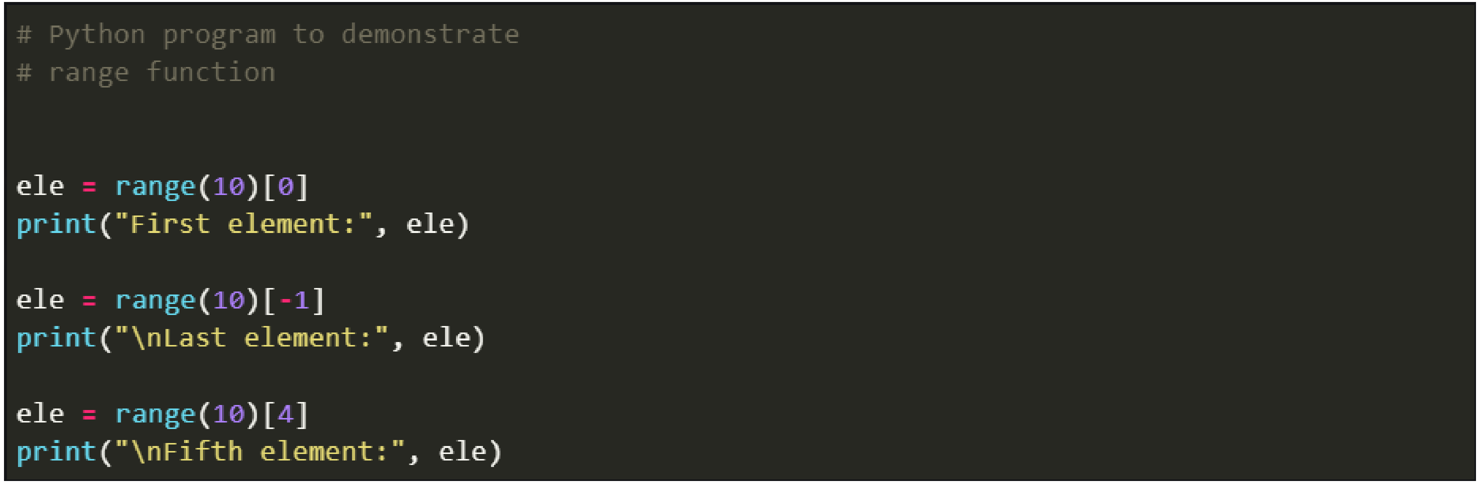


1	3	5	7	9

Start value

Stop value (not included)

Accessing range() with index value

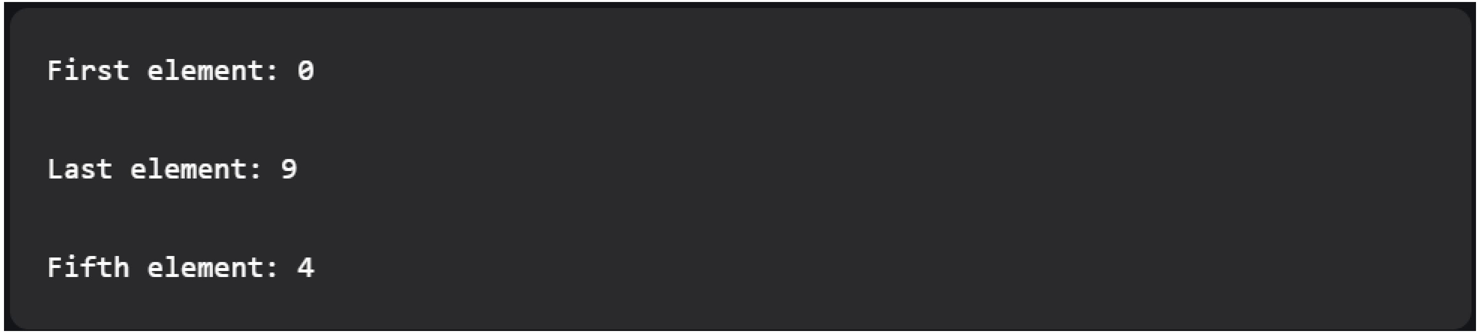


```python
# Python program to demonstrate
# range function


ele = range(10)[0]
print("First element:", ele)

ele = range(10)[-1]
print("\nLast element:", ele)

ele = range(10)[4]
print("\nFifth element:", ele)
```



```
First element: 0


Last element: 9


Fifth element: 4
```

02

Loops

Why using loop?

When programming, we can see there are a lot actions that need to be repeated, typing them all will be a waste of time and resources

By utilizing loop, we can repeat the action many time with little code

Definite iteration

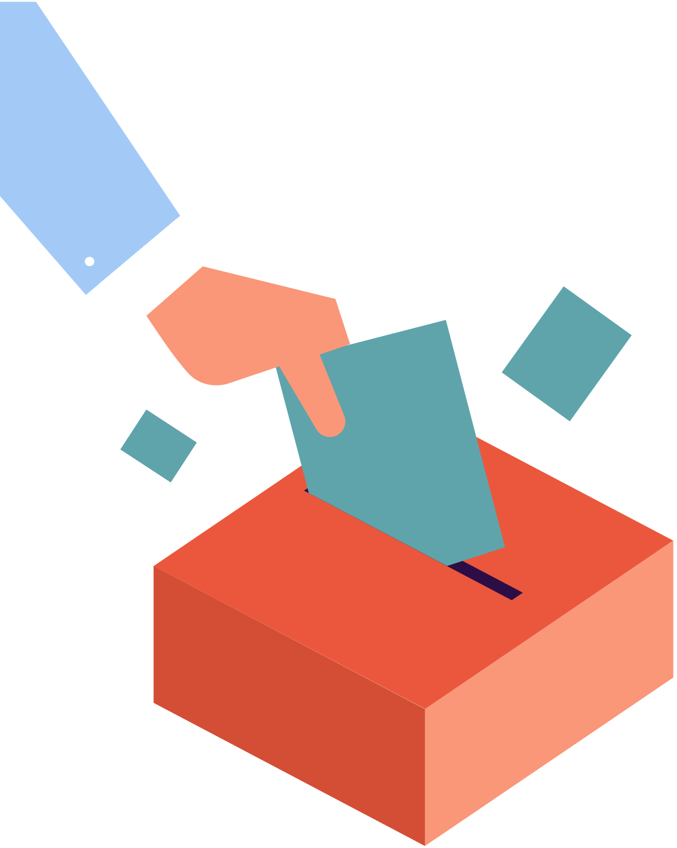
The number of times the designated block will be executed is specified explicitly at the time the loop starts

For loop

Infinite iteration

The number of times the loop is executed isn't specified explicitly in advance. Rather, the designated block is executed repeatedly as long as some condition is met.

While loop



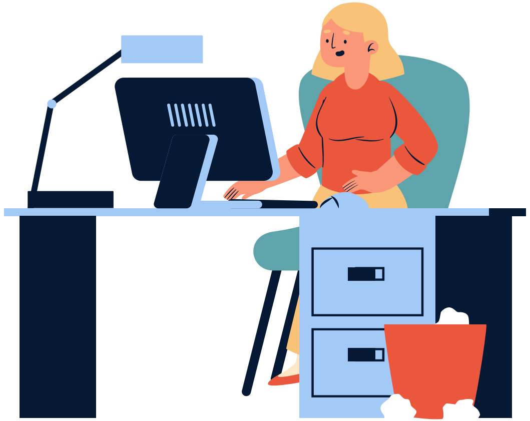
03

For loop

The for loop

Python for loop follows the syntax:

```
for var_name in <collection>:
        <statements>
```



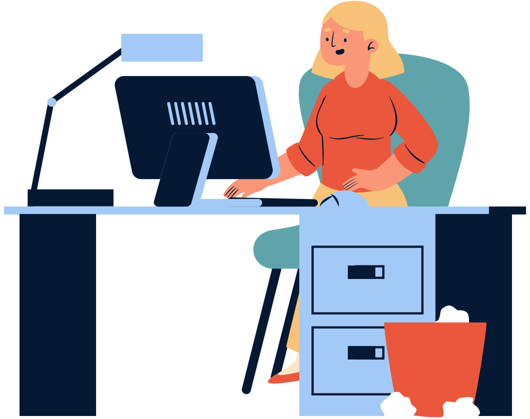
04

While loop

The while loop

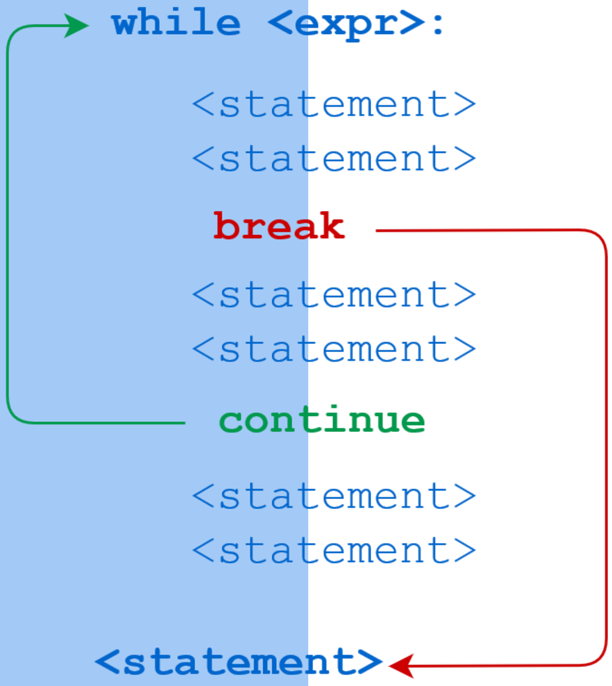
The format of a while loop:

```
while <expression>:
    <statement(s)>
```



continue

immediately terminates the current loop iteration. Execution jumps to the top of the loop



```
while <expr>:
    <statement>
    <statement>
     break
    <statement>
    <statement>
     continue
    <statement>
    <statement>
<statement>
```

break

Immediately terminates a loop entirely. Program execution proceeds to the first statement following the loop body.

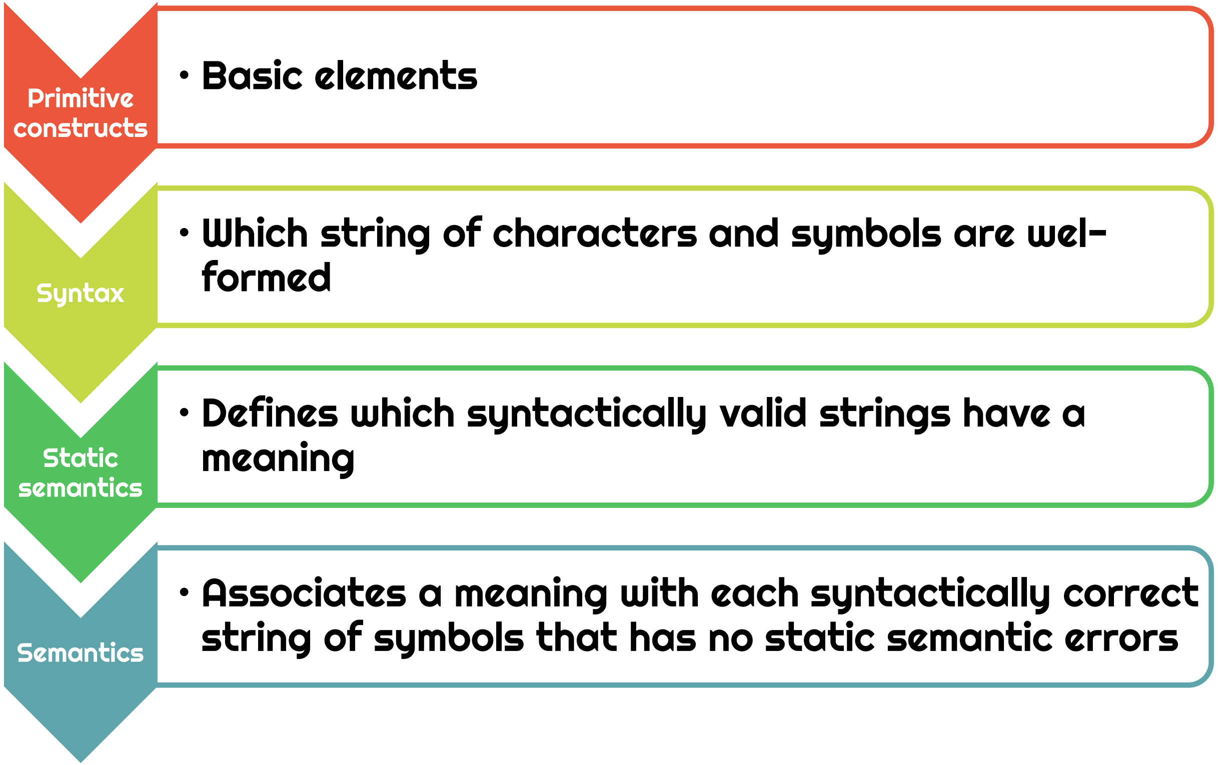
For loops vs While loops

For loops	While loops
➢ Know number of iterations	➢ Unbounded number of iterations
➢ Can end early via *break*	➢ Can end early via *break*
➢ Uses a counter	➢ Can use a counter but must initialize before loop and increment it inside loop
➢ Can rewrite a *for* loop using a *while* loop	➢ May not be able to rewrite a *while* loop using a *for* loop

05

Debugging techniques

Aspect of Language



Primitive constructs

- Basic elements

Syntax

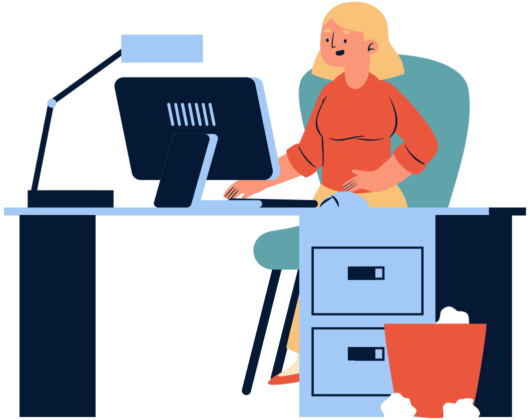
- Which string of characters and symbols are wel-formed

Static semantics

- Defines which syntactically valid strings have a meaning

Semantics

- Associates a meaning with each syntactically correct string of symbols that has no static semantic errors



English

Primitive constructs

- Words

Syntax

- “Cat dog boy”
- => <noun> <noun> <noun>

Static semantics

- “I are big”
- => <pronoun> <linking verb> <adjective>

Semantics

- Ambinguous:
- “Look at the dog with one eye”

Python

Primitive constructs

- Number: 10, 3.2
- Strings: “abcd”
- Simple operators: +, /

Syntax

- Expression: <object> <operator> <object>
- a = 10
- b 52

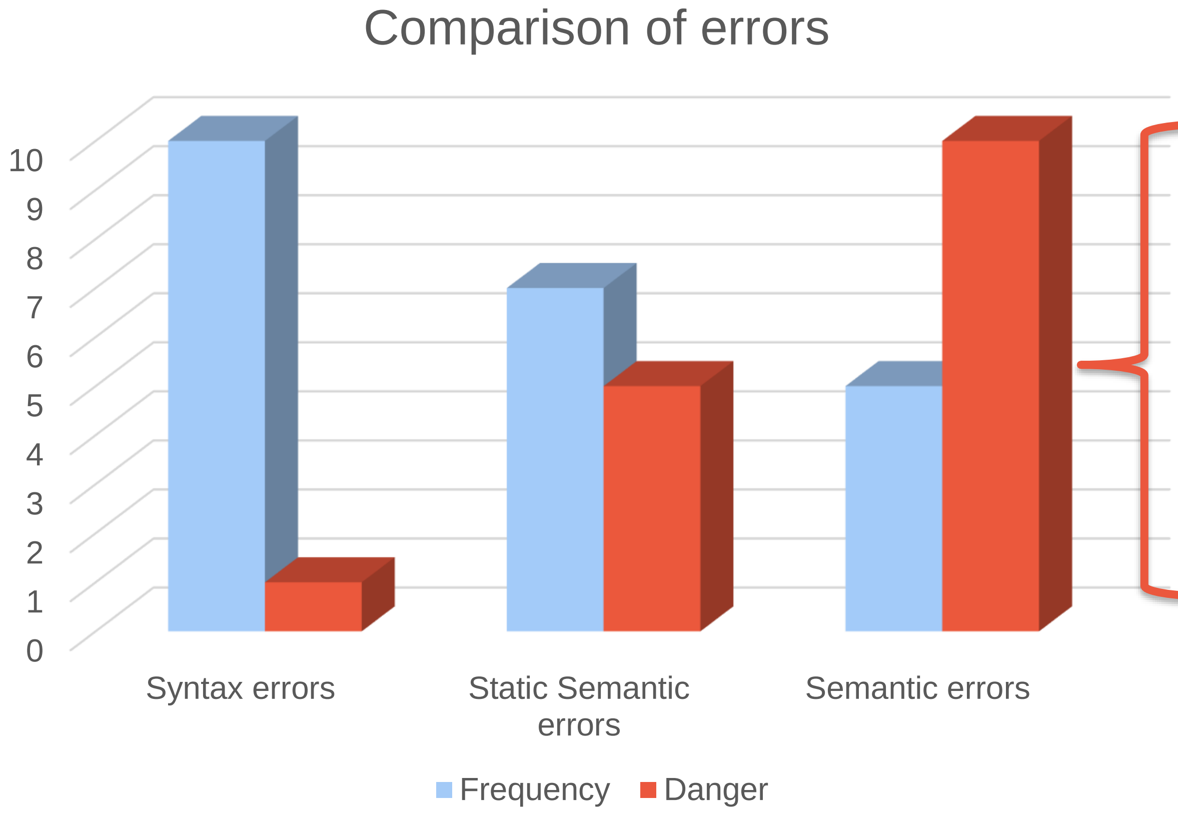
Static semantics

- 3.2/’abc’

Semantics

- There is only one meaning

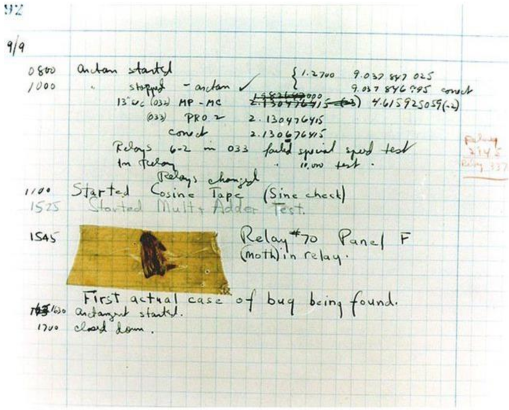
Errors



- ❑ **Program crashes, stops running**
- ❑ **Program runs forever**
- ❑ **Program gives an answer but different than expected**

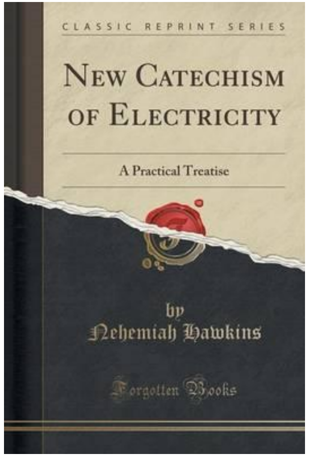
Debug story

- Some have claimed that the discovery of that unfortunate moth trapped in the Mark II led to the use of the phrase debugging. However, the wording, “First actual case of a bug being found,” suggests that a less literal interpretation of the phrase was already common. Grace Murray Hopper, a leader of the Mark II project, made it clear that the term “bug” was already in wide use to describe problems with electronic systems during World War II.
- And well prior to that, Hawkins’ New Catechism of Electricity, an 1896 electrical handbook, included the entry, “The term ‘bug’ is used to a limited extent to designate any fault or trouble in the connections or working of electric apparatus.”
- In English usage, the word “bugbear” means “anything causing seemingly needless or excessive fear or anxiety.”
- Shakespeare seems to have shortened this to “bug,” when he had Hamlet kvetch about “bugs and goblins in my life.”



```
92
9/9
0800  Arctan started
1000    "   stopped - arctan ✓        {1.2700   9.037 847 025
                                               9.037 846 995 correct
        13" uc (032) MP-MC  ~~2.130476415~~ (033) 4.615925059(-2)
             (033) PRO 2    2.130476415
                   correct  2.130676415
        Relays 6-2 in 033 failed special speed test
        in relay        "   11.000 test.                Relay
                Relays changed                          2145
1100  Started Cosine Tape (Sine check)                 Relay 337
1525  Started Mult + Adder Test.
1545                        Relay #70 Panel F
                            (moth) in relay.
      First actual case of bug being found.
1630  Arctangent started.
1700  closed down.
```

The photo of a September 9, 1947, page in a laboratory book from the group working on the Mark II Aiken Relay Calculator at Harvard University.



CLASSIC REPRINT SERIES

NEW CATECHISM OF ELECTRICITY

A Practical Treatise

by Nehemiah Hawkins

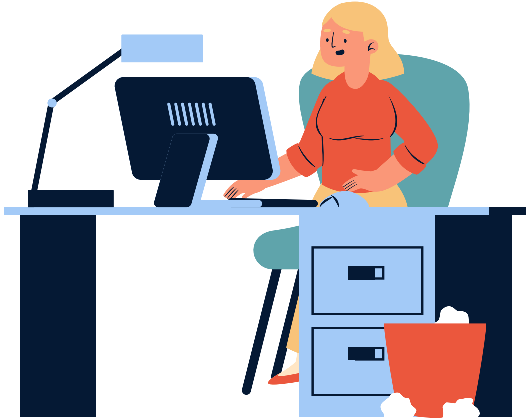
Forgotten Books

Debug

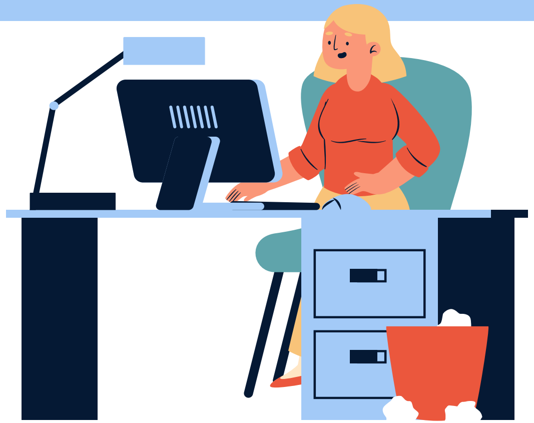
- Debugging starts when testing has demonstrated that the program behaves in undesirable ways.

- Debugging is the process of searching for an explanation of that behavior.

- The key to being consistently good at debugging is being systematic in conducting that search.



Debug steps



Studying available data (test results and program text)

- Try to understand why one test worked and another did not.
- When looking at the program text, keep in mind that you don't completely understand it.

Forming a hypothesis that you believe to be consistent with all the data.

- Narrow: *"If I change line 402 from x < y to x <= y, the problem will go away"*
- Broad: *"my program is not terminating because I have the wrong exit condition in some while loop."*

Design and run a repeatable experiment with the potential to refute the hypothesis

- Put a "print" statement before and after each while loop. If these are always paired, then the hypothesis that a while loop is causing nontermination has been refuted.
- Decide before running the experiment how you would interpret various possible results.

Keep a record of what experiments you have tried.

- "Insanity is doing the same thing, over and over again, but expecting different results"

When the Going Gets Tough

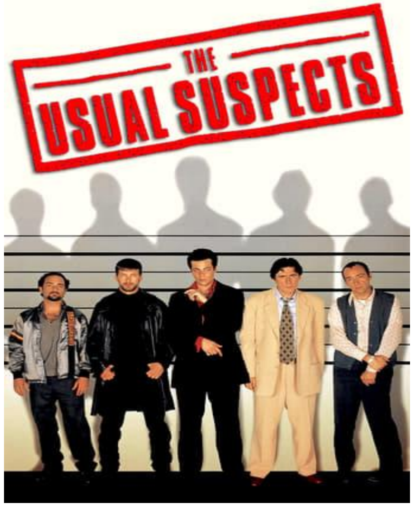
1. Look for the usual suspect

-> *Misspelled a name (lowercase/uppercase...)*

-> *Failed to reinitialize a variable*

-> *Forgotten that some built-in function has a side effect*

...



2. Stop asking yourself why the program isn't doing what you want it to. Instead, ask yourself why it is doing what it is.

When the Going Gets Tough

3. Keep in mind that the bug is probably not where you think it is.

 -> *If it were, you would probably have found it long ago*

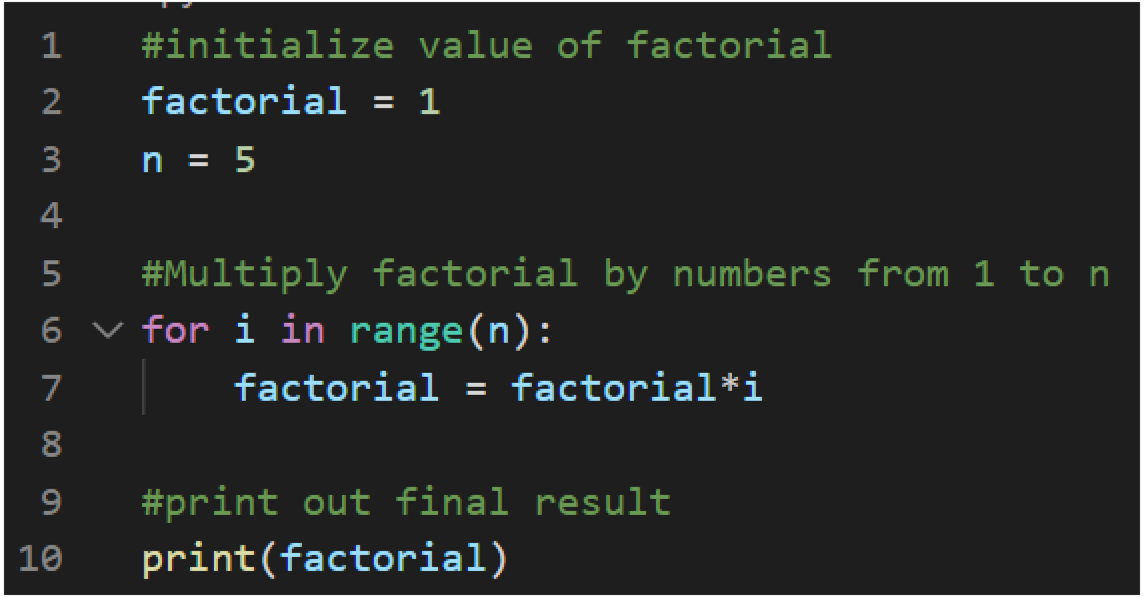
 -> *Sherlock Homes said: "Eliminate all other factors, and the one which remain must be the truth."*

4. Try to explain the problem to somebody else

 -> *Rubber duck debugging*

5. Walk away and try again tomorrow

Visual Studio Code debugger



```
#initialize value of factorial
factorial = 1
n = 5

#Multiply factorial by numbers from 1 to n
for i in range(n):
    factorial = factorial*i

#print out final result
print(factorial)
```

Find the factorial of a number n

THANKS!

See you in the next lesson!

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**.