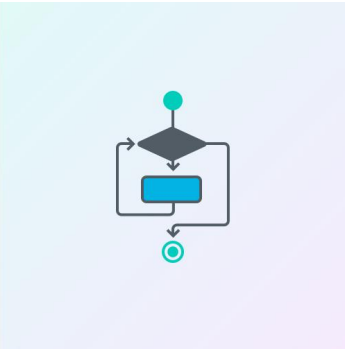
Programming AI in Python

Control Flow



- Conditional Statements
- Boolean Expressions
- For and While Loops
- Break and Continue
- Zip and Enumerate
- List Comprehensions

If Statement

If Statement

An `if` statement is a conditional statement that runs or skips code based on whether a condition is true or false. Here's a simple example.

```
if phone_balance < 5:
    phone_balance += 10
    bank_balance -= 10
```

If, Elif, Else

In addition to the `if` clause, there are two other optional clauses often used with an `if` statement. For example:

```
if season == 'spring':
    print('plant the garden!')
elif season == 'summer':
    print('water the garden!')
elif season == 'fall':
    print('harvest the garden!')
elif season == 'winter':
    print('stay indoors!')
else:
    print('unrecognized season')
```

Indentation

Some other languages use braces to show where blocks of code begin and end. In Python we use indentation to enclose blocks of code. For example, `if` statements use indentation to tell Python what code is inside and outside of different clauses.

In Python, indents conventionally come in multiples of four spaces. Be strict about following this convention, because changing the indentation can completely change the meaning of the code. If you are working on a team of Python programmers, it's important that everyone follows the same indentation convention!

Spaces or Tabs?

The **Python Style Guide** recommends using 4 spaces to indent, rather than using a tab. Whichever you use, be aware that "Python 3 disallows mixing the use of tabs and spaces for indentation."

Python Style Guide

Test 1

In [5]:
```
points = 174  # use this input to make your submission

# write your if statement here
if points >= 1 and points <= 50:
    result = "Congratulations! You won a wooden rabit"
elif points >= 51 and points <= 150:
    result = "no prize"
elif points >= 151 and points <= 180:
    result = "Congratulations! You won a wafer-thin mint"
elif points >= 181 and points <= 200:
    result = "Congratulations! You won a penguin"
else:
    result = "not a valid option"

print(result)
```

```
Congratulations! You won a wafer-thin mint
```

Building Dictionaries
Method 1: Use For Loop

Method 1: Using a `for` loop to create a set of counters

Let's start with a list containing the words in a series of book titles:

```
book_title =  ['great', 'expectations','the', 'adventures', 'of', 'sherl
```

Step 1: Create an empty dictionary.

```
word_counter = {}
```

Step 2. Iterate through each element in the list. If an element is already included in the dictionary, add 1 to its value. If not, add the element to the dictionary and set its value to 1.

```
for word in book_title: if word not in word_counter:
word_counter[word] = 1 else: word_counter[word] += 1
```

Method 2: Using the `get` method

We will use the same list for this example:

```
book_title =  ['great', 'expectations','the', 'adventures', 'of', 'sherloc
```

Step 1: Create an empty dictionary.

```
word_counter = {}
```

Step 2. Iterate through each element, `get()` its value in the dictionary, and add 1.

Recall that the dictionary `get` method is another way to retrieve the value of a key in a dictionary. Except unlike indexing, this will return a default value if the key is not found. If unspecified, this default value is set to None. We can use `get` with a default value of 0 to simplify the code from the first method above.

```
for word in book_title:
    word_counter[word] = word_counter.get(word, 0) + 1
```

Iterating Through Dictionaries with For Loops

Iterating through it in the usual way with a `for` loop would give you just the keys, as shown below:

```
for key in cast:
    print(key)
```

This outputs:

```
Jerry Seinfeld
Julia Louis-Dreyfus
Jason Alexander
Michael Richards
```

If you wish to iterate through both keys and values, you can use the built-in method `items` like this:

```
for key, value in cast.items():
    print("Actor: {}    Role: {}".format(key, value))
```

This outputs:

```
Actor: Jerry Seinfeld    Role: Jerry Seinfeld
Actor: Julia Louis-Dreyfus    Role: Elaine Benes
Actor: Jason Alexander    Role: George Costanza
Actor: Michael Richards    Role: Cosmo Kramer
```

Iterating Through Dictionaries

Run and experiment with this example in the code cell below!

In [ ]:
```
cast = {
            "Jerry Seinfeld": "Jerry Seinfeld",
            "Julia Louis-Dreyfus": "Elaine Benes",
            "Jason Alexander": "George Costanza",
            "Michael Richards": "Cosmo Kramer"
        }

print("Iterating through keys:")
for key in cast:
    print(key)

print("\nIterating through keys and values:")
for key, value in cast.items():
    print("Actor: {}    Role: {}".format(key, value))
```

An image showing the Iterating Through Dictionaries section in the workspace

While Loops

`While` Loops

`For` loops are an example of "definite iteration" meaning that the loop's body is run a predefined number of times. This differs from "indefinite iteration" which is when a loop repeats an unknown number of times and ends when some condition is met, which is what happens in a `while` loop. Here's an example of a `while` loop.

```
card_deck = [4, 11, 8, 5, 13, 2, 8, 10]
hand = []

## adds the last element of the card_deck list to the hand list
## until the values in hand add up to 17 or more
while sum(hand)  < 17:
    hand.append(card_deck.pop())
```

Break, Continue

Sometimes we need more control over when a loop should end, or skip an iteration. In these cases, we use the `break` and `continue` keywords, which can be used in both `for` and `while` loops.

- `break` terminates a loop
- `continue` skips one iteration of a loop

Watch the video and experiment with the examples below to see how these can be helpful.

Zip and Enumerate

`zip` and `enumerate` are useful built-in functions that can come in handy when dealing with loops.

Zip

`zip` returns an iterator that combines multiple iterables into one sequence of tuples. Each tuple contains the elements in that position from all the iterables. For example, printing

`list(zip(['a', 'b', 'c'], [1, 2, 3]))` would output `[('a', 1), ('b', 2), ('c', 3)]`.

Like we did for `range()` we need to convert it to a list or iterate through it with a loop to see the elements.

You could unpack each tuple in a `for` loop like this.

```
letters = ['a', 'b', 'c']
nums = [1, 2, 3]

for letter, num in zip(letters, nums):
    print("{}: {}".format(letter, num))
```